PRODUCT ASSURANCE POLICIES AND PROCEDURES FOR FLIGHT DYNAMICS SOFTWARE DEVELOPMENT

MARCH 1987

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SOFTWARE ENGINEERING LABORATORY SERIES

SEL-87-001

PRODUCT ASSURANCE POLICIES AND PROCEDURES FOR FLIGHT DYNAMICS SOFTWARE DEVELOPMENT

MARCH 1987

NASA

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/ Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

NASA/GSFC (Systems Development and Analysis Branch)
The University of Maryland (Computer Sciences Department)
Computer Sciences Corporation (Flight Systems Operation)

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document. A version of this document was also issued as Computer Sciences Corporation document CSC/TM-87/6017.

The contributors to this document include

Sandra Perry, Leon Jordan, William Decker, Gerald Page (Computer Sciences Corporation)
Frank McGarry, Jon Valett (Goddard Space Flight Center)

Single copies of this document can be obtained by writing to

Frank E. McGarry
Code 552
NASA/GSFC
Greenbelt, Maryland 20771

ABSTRACT

The product assurance policies and procedures necessary to support flight dynamics software development projects for Goddard Space Flight Center are presented. The quality assurance and configuration management methods and tools for each phase of the software development life cycle are described, from requirements analysis through acceptance testing; maintenance and operation are not addressed.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION 1 - INTRODUCTION

This document describes the product assurance policies and procedures necessary to provide quality products throughout the software development life cycle. It is intended for use in flight dynamics software development projects at Goddard Space Flight Center. For this document, product assurance consists of quality assurance and configuration management, both of which are essential for producing maintainable, complete, and reliable software product deliverables.

Quality assurance is the process of reviewing all software development products to ensure that they meet or exceed a predefined set of guidelines and standards and are complete in content. It involves selecting guidelines and standards for the requirements, design, test, system description, user documentation, and code and checking that they are met.

Configuration management is the process of providing an orderly, systematic, and controlled flow of software development products in a changing environment. It involves establishing product baselines and tracking changes to the baselines.

This document is intended for use by flight dynamics software development technical managers, task leaders, and quality assurance and configuration management personnel. A technical manager is defined as the section manager or first-line supervisor responsible for the project.

The purpose of this document is to define the policies and procedures for product assurance and the actions that will be taken. It also provides guidelines for developing quality assurance and configuration management plans. Each technical manager will tailor the plans to the specific project at hand.

This document is a high-level presentation, not an independent self-reference, and should be used in conjunction with the following publications:

- ***Recommended Approach to Software Development*** (Reference 1), for the software development life-cycle phases and the activities and products resulting from each phase
- ***Manager's Handbook for Software Development*** (Reference 2), for planning guidance, standard document formats, and project monitoring guidance
- ***Programmer's Handbook for Flight Dynamics Software Development*** (Reference 3), for programming standards and conventions
- ***Software Verification and Testing*** (Reference 4), for testing procedures and code-reading techniques

The document is structured as follows. Section 2 is an overview of product assurance and its role in the life-cycle management of software development projects. It presents the major goals of quality assurance and configuration management, describes general planning factors, and discusses product assurance products and processes.

Sections 3 and 4 describe the policies and procedures for quality assurance and configuration management, respectively. They include the objectives and approach, methods and tools to be used, activities required during each phase of the software development life cycle, and management considerations.

Appendixes A and B present guidelines for preparing quality assurance and configuration management plans, respectively. Appendix C contains quality assurance checklists, and Appendix D contains standard configuration management forms.

SECTION 2 - PRODUCT ASSURANCE OVERVIEW

2.1 PRODUCT ASSURANCE GOALS

The goal of product assurance is to produce maintainable, complete, and reliable software product deliverables through a combination of quality assurance and configuration management activities. Quality assurance addresses compliance with standards during the production of code and supporting documentation. Configuration management addresses the management of change throughout the software development process.

2.1.1 QUALITY ASSURANCE GOALS

The goal of quality assurance is to ensure the maintainability and completeness of the software product deliverables resulting from the development of flight dynamics software. This is achieved by identifying standards that are to be met or exceeded in the development of the products. These standards are documented in the Software Engineering Laboratory series of documents and cover both the specific content and format for the requirements, design, test, system description and user documentation and the way in which the code will be developed (e.g., top-down, structured, single function per module, uniform comment format). Another essential goal of quality assurance is to ensure that each software product fulfills its specific intent in content. This is accomplished in the normal review process of all project deliverables.

Quality assurance should be made an integral part of software development activities by initiating it early in the software development process. The standards, guidelines, and conventions that establish the ground rules for the development process are identified in the Quality Assurance Plan, normally developed as part of the Software Development/Management Plan. This plan is published and distributed to all development team members, and frequent spot checks and reviews are

made to eliminate doubts regarding the standards and formats to be used throughout the development process.

For the purpose of this document, quality assurance is primarily directed toward the format and content of the software products and the standards used to develop them. Ensuring that the products represent and meet operational requirements is addressed through the normal review of preliminary and final documents and software by the development team members, task leader, and technical manager. Other methods used to ensure that operational requirements are met through the formal design review and testing processes (covered in Reference 4) are not discussed in this document, except for the supporting product assurance activities for the formal reviews.

2.1.2 CONFIGURATION MANAGEMENT GOALS

The goal of configuration management is to produce reliable software product deliverables. This is achieved by providing for an orderly, systematic, and controlled development of the products in a changing environment. It involves identifying products that will be monitored for change management control, establishing the baseline for these products, and maintaining records that trace required changes. Changes are made within a formalized structure under controlled procedures.

The configuration management goals and processes described in this document are concerned with product deliverables outside both the formal Configuration Control Board (CCB) process and hardware configuration management. The interfacing activities with the CCB are, however, included.

2.2 PLANNING

Quality Assurance and Configuration Management Plans are developed at the beginning of each project and are usually included in the Software Development/Management Plan. The plans specifically identify the products to be managed by the

quality assurance and configuration management processes and the tools and methods to be used to carry out the processes. Identifying the individuals who will act as the quality assurer and the configuration manager is also a key factor to be considered in the planning process.

The plans reference the specific standards to be used for a project. Table 2-1 lists the usual software products and the major reference documents containing the standards and formats used in their production. These references are oriented toward software development using the FORTRAN programming language on either IBM or VAX computers. The standards presented in these references will be used unless deviations and substitutions are necessary.

2.2.1 PLANNING FOR LEVEL OF DETAIL

The level of detail expected to be covered in the product assurance process depends on the type of project: operational (e.g., attitude ground support systems), research and development (e.g., tools and analysis systems), or support (e.g., utilities); the criticality of the products; and the amount of effort allocated for product assurance activities. The technical manager's primary guideline is to tailor the level of detail to the specific product.

Common sense must be applied when determining the extent of activities to be conducted for each deliverable. Tradeoffs have to be considered regarding the cost of the product assurance activities and the value added by the activity in relation to the specific product.

Product assurance activities should apply only to product deliverables, that is, not to transitory documents. In the quality assurance process, spot checks of code as it is initially developed provide a less intense review than code-reading techniques; thorough reviews of documents before their

Table 2-1. Standards References

Products	References	
	Recommended Approach to Software Development (Reference 1)	*Manager's Handbook for Software Development* (Reference 2)
Documents		
• Software Development/ Management Plan	• Appendix B	• Section 2
• Requirements Analysis Summary Report	• Appendix B	• Section 4
• Preliminary Design Report	• Appendix B	• Section 4
• Detailed Design Document	• Appendix B	• Section 4
• Test Plans	• Appendix B	• Section 4
• User's Guide	• Appendix B	• Section 4
• System Description	• Appendix B	• Section 4
• Software Development History	• Appendix B	• Section 4
Software	*Programmer's Handbook for Flight Dynamics Software Development* (Reference 3)	
• Prologs	• Appendix F	
• PDL	• Appendix G	
• Code	• Appendix H	
• System Delivery Tape	• Appendix J	
Other Items	*Recommended Approach to Software Development* (Reference 1)	
• PDR Materials	• Section A.2	
• CDR Materials	• Section A.3	

publication ensure their completeness and adherence to standards. Configuration management procedures should be concerned with requirements changes that have major design/performance/resource implications and all changes to controlled software, that is, they should not involve tracking the correction of typographical errors.

2.2.2 PLANNING FOR PROJECT SIZE

The technical manager also tailors the product assurance activities in the product assurance plans to the size of the project. The level of effort or formality of the activities will, however, always depend on the criticality of the software products and not necessarily on the size. General planning guidance for size is as follows:

- Large Projects
  - More than 200K source lines of code
  - Staff of more than 15 people
  - Product assurance activities normally follow formal procedures and are conducted by specialists on a full-time basis. Approximately 5 percent of the total project effort is devoted to product assurance activities.
- Medium Projects
  - 50K to 200K source lines of code
  - Staff of 8 to 15 people
  - A separate, dedicated product assurance staff is normally not necessary, and the responsibility is assigned to one of the technical staff members on a part-time basis. Approximately 2 to 5 percent of the total project effort is devoted to product assurance activities. Attitude ground support system projects normally fall in this category.

- Small Projects
  - Less than 50K source lines of code
  - Fewer than 8 people
  - Product assurance activities are normally accomplished by the task leader on the project and conducted formally or informally depending on the criticality of the project. Support and simulator projects fall into this category.

These levels of effort do not include the effort spent by development team members and the technical manager in performing quality assurance duties that are part of the normal development process. For example, the effort spent by the developers reviewing the code as they develop and unit test or that spent in the normal documentation editing process are not included. This document describes what the product assurance process does include and what must be considered when establishing the level of effort for the project.

2.3 PRODUCTS AND PROCESSES

Figure 2-1 depicts the normal product assurance processes that are applied to software products and supporting documentation throughout the software development life cycle. Some products are placed under product assurance for only one phase of development and are then superceded by other products. For example, the design of the system is documented beginning with the Requirements Analysis Summary Report and is replaced in successive phases of development by the Preliminary Design Report; the Detailed Design Document; and finally by the System Description, which is delivered with the system. Each of these documents is normally placed under configuration management until it is replaced by the next document in the series. Other products, such as the Software Development/Management Plan and the controlled software library, are placed under the

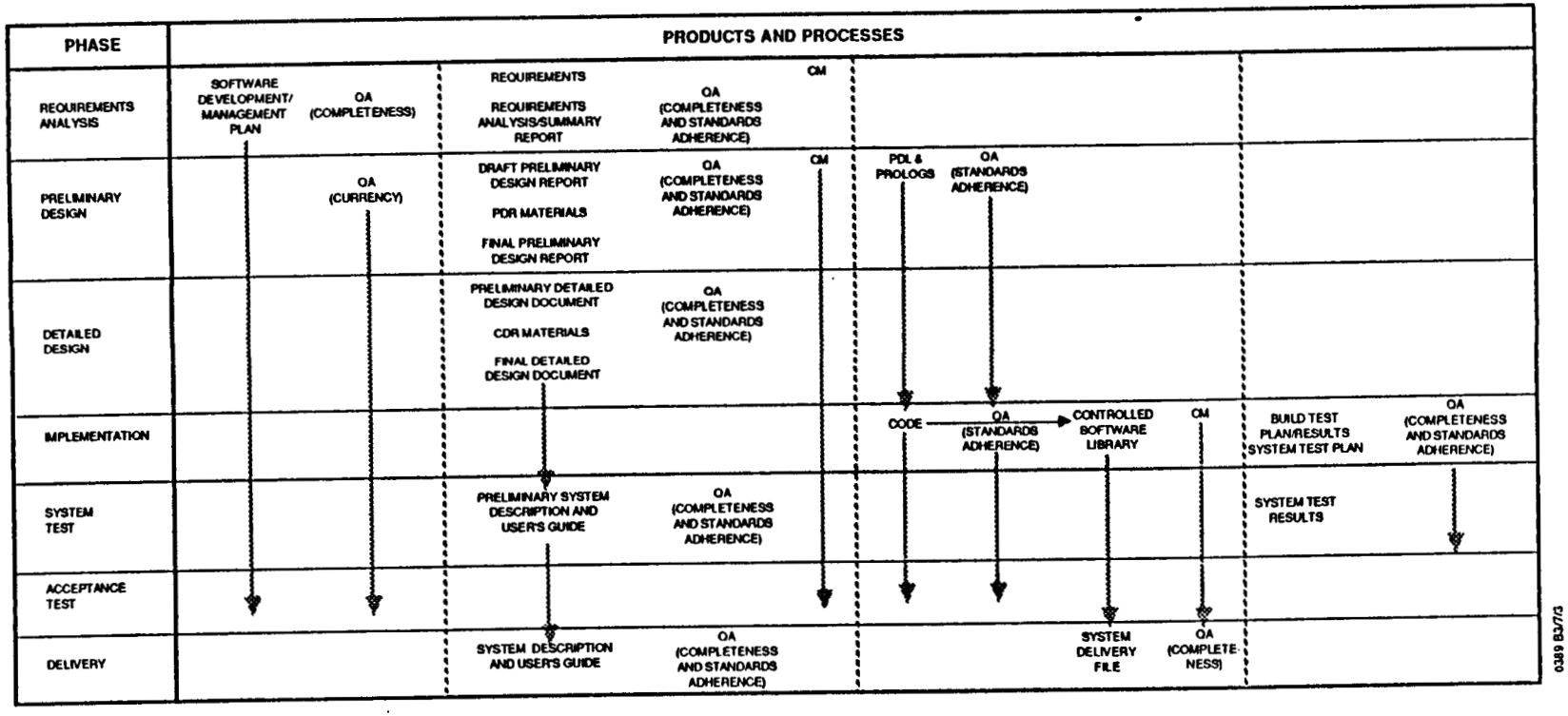

Figure 2-1. Items Typically Part of Product Assurance Process

quality assurance and configuration management processes, respectively, for the duration of the development activity.

To further amplify Figure 2-1, the role of product assurance in the software development life-cycle phases is described briefly in the following subsections. Reference 1 discusses the life-cycle phases in more detail.

2.3.1 REQUIREMENTS ANALYSIS PHASE

- Development Activities--Functional specifications are translated from mission terms into a software-supportable form and documented in the Requirements Analysis Summary Report, which is used as a basis for preliminary design. The final version of the Functional Specifications and Requirements Document is also completed, and a system requirements review (SRR) is held to evaluate the completeness of the requirements. The Software Development/Management Plan is also produced.

- Product Assurance Activities--Quality assurance and configuration management activities are planned and documented in the Software Development/Management Plan. Quality assurance checks for completeness, content, and format are also made on the Software Development/Management Plan and the Requirements Analysis Summary Report.

2.3.2 PRELIMINARY DESIGN PHASE

- Development Activities--A software system architecture is defined based on the requirements given in the Functional Specifications and Requirements Document and the Requirements Analysis Summary Report. This architecture is translated into major functional subsystems and documented in the Preliminary Design Report. The functional design is formally presented for review at the preliminary design review (PDR). Preliminary design is considered complete when responses to the PDR comments and criticisms have been incorporated as change pages into the final Preliminary Design Report.

- Product Assurance Activities--Quality assurance activities include reviews of the Preliminary Design Report and the preliminary design review materials for completeness, and the Software Development/Management Plan for currency. Module PDL and prolog development are also spot checked and reviewed for standards compliance. Configuration management activities involve monitoring the question-and-answer process between the developers and the requirements team, recording PDR changes, and tracking them into the final Preliminary Design Report.

2.3.3 DETAILED DESIGN PHASE

- Development Activities--The system architecture defined in preliminary design is extended to the subroutine level, producing the "code-to" specifications for the system. Design specifications are documented in the Detailed Design Document. The detailed design is formally presented at the critical design review (CDR). The design is considered complete when responses to the CDR comments and criticisms have been incorporated into the Detailed Design Document.

- Product Assurance Activities--Quality assurance activities include spot checks and reviews of the program prologs and PDL, reviews of the Detailed Design Document and the CDR materials for standards compliance and completeness, and a review of the Software Development/Management Plan for currency. Configuration management activities involve tracking requirements changes and questions and answers between the requirements team and developers, recording CDR changes, and tracking the changes into the final Detailed Design Document.

2.3.4 IMPLEMENTATION PHASE (CODE AND UNIT TESTING)

- Development Activities--New modules are coded from the design specifications, old code is revised to meet new requirements, each module is unit tested and integrated into the system, and integration testing is performed. A System Test Plan

is also produced. Implementation is considered complete when all code is integrated into the controlled software library. An independent acceptance test team prepares the Acceptance Test Plan.

- Product Assurance Activities--Quality assurance activities include spot checks and reviews of the code, code reading to check for standards compliance, a review of the System Test Plan for completeness, and a review of the Software Development/Management Plan for currency. Configuration management activities involve tracking questions and answers and requirements changes into the Detailed Design Document. The controlled software library baseline is established at the beginning of this phase, and changes and additions are tracked to it.

2.3.5 SYSTEM TESTING PHASE (SYSTEM INTEGRATION AND TESTING)

- Development Activities--The integrated system is validated, and end-to-end system capabilities are tested according to the System Test Plan. System testing is considered complete when all tests specified in the System Test Plan have been executed successfully. The preliminary User's Guide and System Description documents are also produced.

- Product Assurance Activities--Quality assurance activities involve the review of the preliminary User's Guide and System Description for standards compliance and content and review of the Software Development/Management Plan for currency. Configuration management activities include tracking questions and answers and tracking changes to requirements, the Detailed Design Document, and the controlled software library. Baselines are established for the preliminary User's Guide and System Description.

2.3.6 ACCEPTANCE TESTING PHASE

- Development Activities--An independent acceptance test team tests the system to ensure that the software meets all requirements. After the successful completion of acceptance testing, the final versions of the software and the system documentation are delivered to the customer and, if applicable, an operational readiness review (ORR) is held to evaluate the readiness of the system to support operations.

- Product Assurance Activities--The final quality assurance review ensures that all project deliverables are complete. The controlled software library remains under configuration management control until the end of this phase, when the system delivery file is delivered.

SECTION 3 - QUALITY ASSURANCE

The objective of quality assurance is to ensure that software products and supporting documentation meet or exceed the standards established for flight dynamics software, resulting in maintainable and complete software product deliverables. Early and thorough planning is essential, as is the application of quality assurance procedures early in the development of each new product throughout the development life cycle.

The quality assurance role is shared by everyone on the development team. The responsibilities of the quality assurer are to keep the quality assurance process highly visible to all members of the development team, to ensure that the quality assurance milestones are met, and to maintain a record of the quality assurance activities. The technical manager is responsible for identifying the standards and guidelines to be followed for the project; the technical manager and the task leader are normally responsible for reviewing the documentation for completeness, standards compliance, and content. Team members generally share code-reading tasks. The quality assurer normally conducts frequent spot checks of the code for standards compliance; the quality assurer and the task leader review the code. Figure 3-1 depicts the quality assurance process.

The purpose of this section is to present the policies and procedures for performing quality assurance activities to support flight dynamics software development. It is organized as follows: First, the quality assurance stage is set by presenting management considerations for planning and the quality assurance plan. Second, quality assurance methods are discussed along with the tools that are candidates to support the quality assurance process. Third, the products that are normally quality assured and the processes, that is, the applications of the methods and tools to these products, are ..

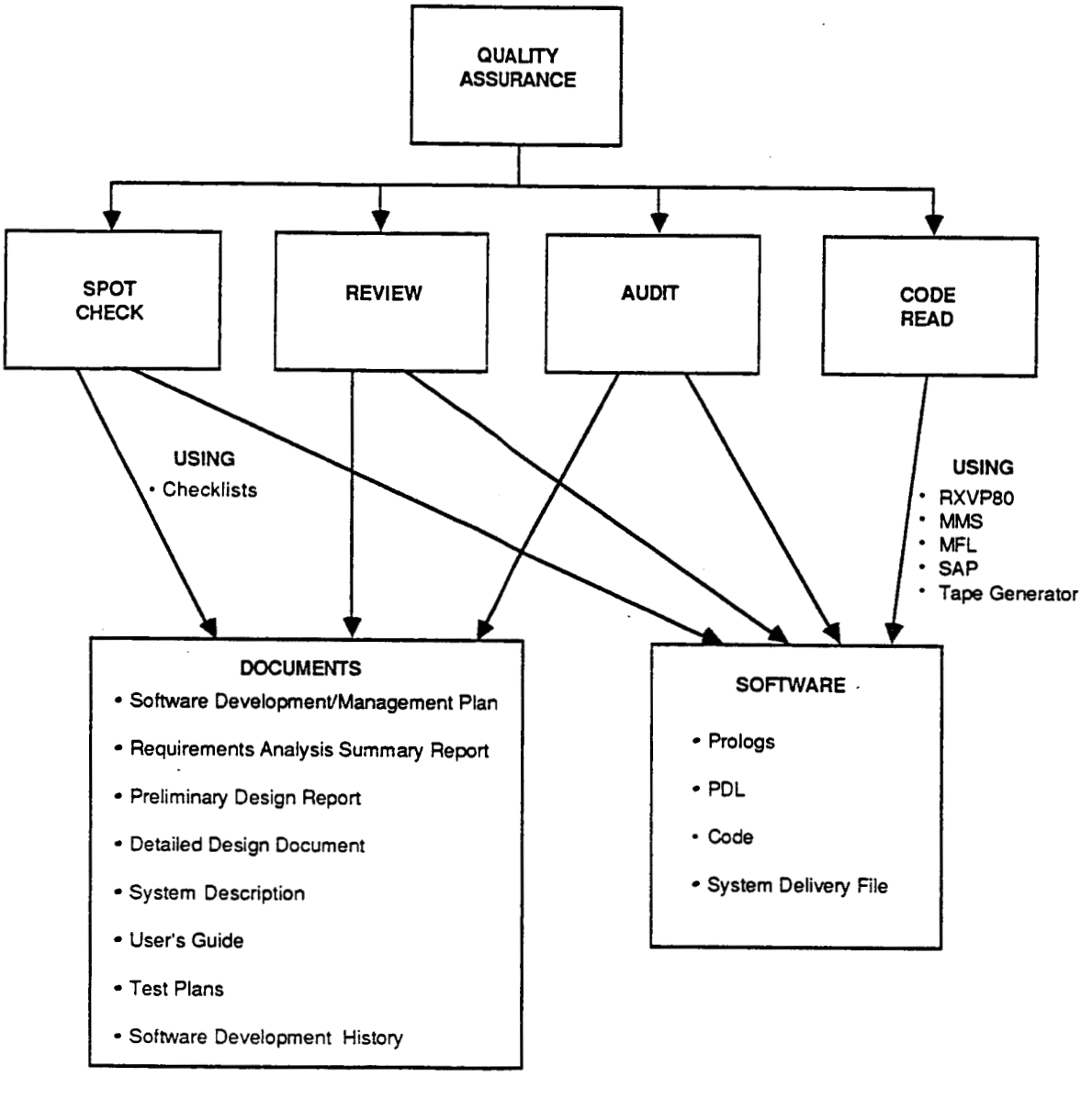

Figure 3-1. The Quality Assurance Process

described. Fourth, all of the previous information is summarized in one page of quality assurance activities for each life-cycle phase of development.

3.1 MANAGEMENT CONSIDERATIONS

3.1.1 PLANNING

Before the Quality Assurance Plan is developed, the amount of effort to be allocated for quality assurance activities must be determined. Planning considerations for the amount of effort required for product assurance activities are discussed in Section 2.2. Another major decision is the selection of the person (or persons) responsible for the quality assurance process. This individual should be knowledgeable in quality assurance methods, tools, policies, and procedures and have a strong propensity to follow up as a matter of practice. The quality assurer must be someone who works well with the technical manager and who can be relied on to work independently to ensure that the quality assurance tasks are accomplished.

3.1.2 QUALITY ASSURANCE PLAN

The Quality Assurance Plan specifies the procedures to be applied during each phase of the software development life cycle and assigns responsibility for quality assurance. This plan is normally developed as part of the Software Development/ Management Plan. The technical manager is responsible for this plan and tailors it to the specific project. The plan addresses the following:

- Standards and conventions to be followed by the development team
- Formats for document deliverables (References 1 and 2) tailored to the specific project
- Checklists (Appendix C) for evaluating standards compliance and product completeness

- Deviations from the policies and procedures as stated in this document and the rationale for these deviations
- Specific products to be monitored by quality assurance procedures
- Methodologies and tools to be used for each product
- Quality assurance milestones based on the major milestones of the project and the product deliverables (Compliance with standards must be enforced in the early stages of software and document production. Spot checks and reviews for compliance must therefore be scheduled appropriately.)
- Person (or persons) responsible for quality assurance of the project and those who will review and audit the products

Appendix A presents guidelines for preparing a Quality Assurance Plan.

3.2 QUALITY ASSURANCE METHODS AND TOOLS

3.2.1 QUALITY ASSURANCE METHODS

The primary methods used in the quality assurance process are as follows:

- Spot checks
- Reviews
- Audits
- Code reading

Spot checks are short-notice reviews of software and documentation samples to monitor standards compliance. Spot checks are normally made at the beginning of product development, that is, when a document is being drafted or when coding begins. The quality assurer performs short-notice spot checks

with each developer in the first few weeks of each new development activity and provides the developers with specific feedback on standards compliance. Followup spot checks are made with developers who have not complied with the standards. Spot checks are made frequently, but normally on a limited amount of material and on an informal basis.

Reviews are a formal process and are normally scheduled at major milestones throughout the development cycle. The products specified for review are reviewed and certified for standards compliance, completeness, and content. Reviews are normally made by the technical manager, the task leader, and the quality assurer. The quality assurer records the completion of the reviews to establish traceability.

Audits are a formal review used for ensuring that proper practices and procedures are being followed, for projects in trouble, and for critical products where there is no room for failure. Audits are normally performed by personnel independent of the development activity or possibly from another organization. Audits of the software products have not been specified in the procedural material that follows. The technical manager specifies the use of audits in the Software Development/Management Plan when they are appropriate. Reference 2 (Section 7) describes the audit procedure.

Code reading is a systematic procedure for reading the program to determine if it is correct with respect to its intended function and logic. During this process, the code reader also checks to see if the code complies with established standards. The code-reading method is used after the code has been developed and compiled and before unit testing. Development team members are responsible for code reading, and the quality assurer is responsible for recording that code reading has been performed for each program.

Specific forms for recording actions and certification forms for signoff of a finished product are used and maintained by the quality assurer. Traceability records of quality assurance activities are also retained in a notebook.

3.2.2 QUALITY ASSURANCE TOOLS

The tools available for use in the quality assurance process are as follows:

- Checklists
- RXVP80
- LOADMAP
- FORTRAN Static Source Code Analyzer Program (SAP)

The following tools are primarily used for configuration management but are included in this section because they can be used to determine the completeness of the software components:

- Module Management System (MMS)
- Master File List (MFL)

3.2.2.1 Checklists

Checklists contain the list of items that must either be accomplished or considered for successful completion of an activity. Appendix C contains the recommended quality assurance checklists, those normally used for standard development activities. Technical managers will develop appropriate checklists for nonstandard development efforts that do not conform to the normal life-cycle phases or that require different products than are specified in this document.

3.2.2.2 RXVP80

RXVP80 is a tool available on both the VAX and IBM computers for structural analysis of FORTRAN source code. It is applied to each program module and is used in conjunction with code reading to detect logical inconsistencies in the code. For example, it flags inconsistencies in data modes and types in

expressions, assignments, and calling sequences. It also checks for a consistent number of arguments in calling sequences and identifies the use of variables before they are set. Reference 5 describes RXVP80 in more detail.

RXVP80 is used during the implementation phase to determine inconsistencies in individual modules after they have been unit tested and before they are placed in the controlled software library. It is also used to determine integration inconsistencies after the modules have been added to the library. RXVP80 runs are normally made for groups of modules after unit testing and on a periodic basis from the controlled software library. The frequency of runs is determined by the amount of library activity. Weekly or biweekly runs are the norm.

The developers normally make RXVP80 runs on their own code from their individual libraries, and the configuration manager normally makes the runs from the controlled library. The quality assurer keeps track of the RXVP80 runs for each module in the controlled software library.

3.2.2.3 LOADMAP

LOADMAP is a quality assurance utility available on the IBM computer that provides the following information for execution modules: a linkage-editor map, an alphabetic program unit list, an unreferenced program unit list, and cross-references of calls and references for all program units. LOADMAP can be used for completeness and consistency checks for execution modules. It is normally used every time a new or changed execution module is created and is used by the individual creating the execution module to quality assure it.

3.2.2.4 FORTRAN Static Source Code Analyzer Program

SAP is a tool available on both the VAX and IBM computers for use during the implementation phase to analyze the code for

complexity. It should be used on groups of modules, for example, on all modules of a subsystem to identify module outliers in terms of complexity. For example, the number of compound statements or the number of nesting levels within the group of modules can be compared. The developer can use this information to determine if there are modules that exceed the norm and can then analyze these modules to see if a more direct coding approach can be taken. Reference 6 describes SAP in more detail.

The quality assurer keeps track of SAP usage for all program modules. The configuration manager normally runs SAP because the runs are made using the controlled software library(s).

3.2.2.5 Module Management System

MMS is a tool available on the VAX computer to build system execution modules. It provides descriptions of all components of the execution module and controls compiling, assembling, and creating object modules and generating an execution module. It obtains information specifying the lineage of each system component by examining the components' creation dates. System consistency can be assured based on the dates. For example, the execution module will automatically be recreated using new object code created at a later date than the previous execution module. MMS is used primarily as a tool for configuration management, but it is also used to quality assure the execution modules and the system delivery file for completeness. Reference 7 describes MMS in more detail.

MMS is used during the implementation phase when execution modules are initially created for integration testing. It is also used throughout the succeeding testing phases and for the system delivery file. The configuration manager will normally make the MMS runs because the basis for MMS is the source code residing in the controlled software library(s).

3.2.2.6 Master File List and Tape Generator

The MFL, which is available on the VAX computer, is also primarily used for configuration management but can serve as a quality assurance completeness check for all files in the controlled software library. The MFL can be compared to previous lists to determine differences and completeness. Reference 3 (Appendix J) discusses the MFL in more detail.

The MFL is created when the controlled software library is first created and is updated through subsequent phases of development until the system delivery file is delivered. It is normally reviewed by the configuration manager on a periodic basis, every 1 or 2 weeks, for example, and the completeness check is made by the quality assurer or the configuration manager, whoever is specified.

The tape generator is used with the MFL to create a system delivery tape automatically from the MFL at the end of the development cycle. Reference 3 (Appendix J) discusses the tape generator capability in more detail. Using both the MFL and the tape generator ensures a complete system delivery tape.

3.3 QUALITY ASSURANCE PRODUCTS AND PROCESSES

The primary products monitored in the quality assurance process are documents, supporting materials (e.g., PDR and CDR materials), and software.

3.3.1 DOCUMENTS

The documents monitored in the quality assurance process are as follows:

- Software Development/Management Plan
- Requirements Analysis Summary Report
- Preliminary Design Report
- Detailed Design Document
- User's Guide
- System Description

- Build Test Plan and Test Results
- System Test Plan and Test Results
- Software Development History

The Acceptance Test Plan is not discussed in this document because it is normally prepared by the requirements team; it is not the responsibility of the software developers. Most of the documents noted above will be placed under configuration management for change management control.

The User's Guide and the System Description are produced according to the same schedule and, in this document, are discussed together. (In some systems, they may be combined into one document.)

Quality assurance reviews of the Software Development/ Management Plan are scheduled periodically to ensure that this document is current. It is updated throughout the life cycle, particularly at major milestones or when events trigger a change in the management approach. Formal configuration management procedures are not recommended for this document because it is the manager's tool, not a project deliverable. The initial issue of this document is reviewed for completeness according to the standard formats presented in References 1 and 2. The technical manager is responsible for this plan.

Quality assurance reviews of the Build and System Test Results are made for completeness to ensure that all test results and any discrepancies are documented.

The following procedures are used for quality assuring all of the other documents:

1. The quality assurer distributes the standard format for the document (the standard formats are given in References 1 and 2) to team members along with the design, coding, and test plan development assignments.

2. The quality assurer makes frequent spot checks early in the development of each document to determine if the standard is being followed and to clarify any questions regarding format. Problems that cannot be resolved between the quality assurer and the developer are brought to the attention of the task leader or the technical manager for resolution, depending on the nature of the problem.

3. The document is reviewed using the standard format and checklist before publication or distribution as follows: The task leader reviews the document for content, format, consistency, and grammar and works with the individual drafters to clarify or redo their specific areas of responsibility. This task may be shared with the technical manager, or the technical manager may review the document after the task leader has made his/her review. Formal document deliverables are normally passed to a technical publications staff for editing and formatting for publication. The technical manager is responsible for determining that the substance of the document has not been altered in this formal editing process.

4. The technical manager and the quality assurer sign the document, thus certifying that it adheres to the standard and that the content is as expected.

5. The quality assurer records in the quality assurance notebook that the document has been certified as meeting the standard.

3.3.2 SUPPORTING MATERIALS

The supporting materials monitored in the quality assurance process are as follows:

- PDR materials
- CDR materials

The technical manager, the task leader, and the quality assurer are responsible for reviewing the materials supporting

the PDR and CDR. The materials normally consist of briefing materials and supporting documentation. The recommended standards for these materials are presented in Reference 1 (Appendix A). Reviews are made of the content and format for consistency of presentation and for typographical errors. The quality assurer records that the reviews have been made in the quality assurance notebook.

3.3.3 SOFTWARE

The software products monitored in the quality assurance process are as follows:

- Software prologs, PDL, and code
- System delivery file

The procedures for quality assuring the code are as follows:

1. The quality assurer makes frequent spot checks of the module prologs and PDL during the design phases and of the code (e.g., source code, job control language (JCL), panels, Graphic Executive Support System (GESS) displays) during the implementation phase. These checks are to ensure that the developers are practicing good habits using specified standards. These standards are presented in Reference 3 for FORTRAN development projects. When problems occur that cannot be resolved between the quality assurer and the developer, they are brought to the task leader or the technical manager for resolution, depending on the nature of the problem. The quality assurer records when the spot checks are made in the quality assurance notebook.

2. Development team members perform code reading on each coded module after it is compiled. This is used as a further check for standards compliance as well as for logic and content. The quality assurer records that each module has been code read and by whom.

3. RXVP80, if used, is run for groups of FORTRAN coded modules, normally by the individual developers, before the code is added to the controlled software library.

4. RXVP80, if used, is also run on FORTRAN modules as they are integrated into the controlled software library or libraries. The configuration manager usually makes these runs and passes information regarding them to the quality assurer for recording in the quality assurance notebook. Other quality assurance tools, such as SAP and LOADMAP, may also be run. Running these tools is normally the responsibility of the configuration manager because they are run from the controlled software library. If their use is to be tracked formally by the quality assurer, he/she must be informed that they are being used and must follow up to ensure that the results are used to improve the quality of the software.

5. Periodic reviews are also conducted at project milestones (e.g., at the completion of each build/release of the software), using the standard as found in Reference 3 and the checklists as found in Appendix C, to ensure that all standards have been met. Reviews are normally made by the task leader and are recorded by the quality assurer.

The procedures for quality assuring the system delivery file are as follows: The file is checked for completeness in accordance with the standards presented in the Quality Assurance Plan. Building the system delivery file at the end of the project is normally a team effort of the task leader, quality assurer, and configuration manager. The configuration manager is involved because the system source code resides in the controlled software library. Tools such as MMS, the MFL, and the tape generator can be used, if available, to aid in the completeness test. The task leader is responsible for certifying that the system delivery file is complete, and the quality assurer records this action.

3.4 QUALITY ASSURANCE LIFE-CYCLE-PHASE SUMMARY

The major quality assurance actions for each development phase are summarized in the following subsections. The standards, checklists, methods, and tools to be used for quality assuring each development product (both code and supporting documentation) will be specified in the Quality Assurance Plan of the Software Development/Management Plan. The recommended standards for each product are referenced in Table 2-1.

3.4.1 QUALITY ASSURANCE IN THE REQUIREMENTS ANALYSIS PHASE

(QA=Quality Assurer, TL=Task Leader, TM=Technical Manager)

Products

- Software Development/Management Plan
- Requirements Analysis Summary Report

Actions

- Develop Quality Assurance Plan in Software Development/Management Plan (TM)
- Software Development/Management Plan
  - Quality assure against checklist and standard and certify complete (TM, TL)
  - Record publication date and certification (QA)
- Requirements Analysis Summary Report
  - Review for completeness, content, and standards (TL, TM) and certify (QA)
  - Record publication date and certification in quality assurance notebook (QA)

3.4.2 QUALITY ASSURANCE IN THE PRELIMINARY DESIGN PHASE

(QA=Quality Assurer, TL=Task Leader, TM=Technical Manager)

Products

- Preliminary Design Report
- PDR materials
- Software Development/Management Plan
- Module prologs and PDL

Actions

- Distribute Preliminary Design Report standard formats to all team members (QA) when design assignments are made
- Preliminary Design Report
  - Spot check draft material using checklist and standard, sampling each developer's material (QA)
  - Review drafts for completeness and content using checklist and standard (TM, TL)
  - Certify and record publication date of document in quality assurance notebook (QA)
- Review PDR materials for completeness using standard (TM, TL, QA)
- Software Development/Management Plan
  - Review for currency with PDR results (TM)
  - Review updates, if any, for content (TM)
  - Record update and quality assurance certification (QA)
- Module prologs and PDL
  - Spot check each developer's efforts for standards compliance (QA)
  - Review all at end of phase to ensure they are complete and comply with standards (TL) and record review in quality assurance notebook (QA)

3.4.3 QUALITY ASSURANCE IN THE DETAILED DESIGN PHASE

(QA=Quality Assurer, TL=Task Leader, TM=Technical Manager)

Products

- Detailed Design Document
- CDR materials
- Software Development/Management Plan
- Module prologs and PDL

Actions

- Distribute Detailed Design Document standard formats to all team members (QA) when design assignments are made
- Detailed Design Document
  - Spot check draft material using checklist and standard, sampling each developer's material (QA)
  - Review drafts for completeness and content using checklist and standard (TM, TL)
  - Certify and record publication date of document in quality assurance notebook (QA)
- Review CDR materials for completeness using standard (TM, TL, QA)
- Software Development/Management Plan
  - Review for currency with CDR results (TM)
  - Review updates, if any, for content (TM)
  - Record update and quality assurance certification (QA)
- Module prologs and PDL
  - Spot check each developer's efforts for standards compliance (QA)
  - Review all at end of phase to ensure they are complete and comply with standards (TL) and record review in quality assurance notebook (QA)

3.4.4 QUALITY ASSURANCE IN THE IMPLEMENTATION PHASE

(QA=Quality Assurer, TL=Task Leader, TM=Technical Manager)

Products

- Source code
- Test plans and results
- Draft User's Guide and System Description
- Software Development/Management Plan

Actions

- Publicize and distribute standards for code production, test plans, and User's Guide and System Description (QA)
- Source code
  - Conduct frequent spot checks using checklist and standard, sampling all developers' code; record in quality assurance notebook (QA)
  - Keep track of code reading of each module and use of any quality assurance tools (QA)
  - Review all source code for standards compliance and completeness, normally at completion of each software build/release (TL), and record (QA)
- Test plans and results
  - Review build/release and system test plans for completeness using checklist and standard (TM, TL)
  - Record reviews in quality assurance notebook (QA)
  - Review test results for completeness and documented discrepancies (TL, TM) and record (QA)
- Draft Users's Guide and System Description
  - Review draft material at end of each build/release for standards compliance (TL, TM)
  - Record reviews in quality assurance notebook (QA)
- Software Development/Management Plan
  - Review for currency and update if necessary (TM)
  - Record update and certify (QA)

3.4.5 QUALITY ASSURANCE IN THE SYSTEM TESTING PHASE

(QA=Quality Assurer, TL=Task Leader, TM=Technical Manager)

Products

- System test results
- Preliminary User's Guide and System Description
- Software Development/Management Plan

Actions

- Review system test results for completeness and ensure that all discrepancies are documented (TL, TM) and record (QA)
- Preliminary User's Guide and System Description
  - Review for standards compliance, completeness, and content (TL, TM)
  - Certify and record that document fulfills quality assurance requirements (QA)
- Software Development/Management Plan
  - Review for currency and update if necessary (TM)
  - Record update and certify (QA)

3.4.6 QUALITY ASSURANCE IN THE ACCEPTANCE TESTING PHASE

(QA=Quality Assurer, TL=Task Leader, TM=Technical Manager)

Products

- User's Guide and System Description
- Software Development History
- System delivery file

Actions

- User's Guide and System Description
  - Review for standards compliance, completeness, and content (TL, TM)
  - Certify and record that document fulfills quality assurance requirements (QA)
- Software Development History
  - Review for standards compliance, completeness, and content (TL, TM)
  - Certify and record that document fulfills quality assurance requirements (QA)
- System delivery file
  - Review for completeness using checklist, standard, and available quality assurance tools (TL, QA)
  - Certify that system delivery file is complete and ready for delivery (QA)

SECTION 4 CONFIGURATION MANAGEMENT

The objective of configuration management is to provide a systematic method of controlling, tracking, and incorporating change into software products and supporting documentation throughout the software development life cycle. Change is managed to ensure the integrity, consistency, and reliability of the software products. Changes occur in two different ways: (1) through the natural evolution of the software development activity, in which functional specifications are translated into a preliminary design, further refined into a detailed design, and finally developed into code; as the system grows; and as errors are detected and corrected during the testing phases and (2) as a result of external influences such as a change in the environment or requirements. This latter type of change is normally unplanned, for example, when the planned launch of a satellite from the Space Transportation System is changed to an unmanned vehicle. The payload is reduced, which results in a change to the satellite's flight dynamics characteristics. These changes would dictate major design changes to an attitude ground support system.

Managing evolutionary change is necessary in ensuring that the functionality of the system is consistent with system requirements and faithfully represents those requirements as the development proceeds. Evolutionary change is managed through the following processes: PDR and CDR; system walk-throughs; the interaction of the requirements team and the developers; release and system testing and software error correction; and acceptance testing, which is devoted to testing the functionality of the system against operational requirements. The configuration management procedures described in this document support the control of evolutionary change by tracking

(1) questions and answers between the requirements team and the developers, (2) changes resulting from the PDR and CDR, and (3) changes and additions to the controlled software library.

Managing unplanned, environmentally induced changes involves establishing appropriate product baselines and tracking and controlling changes to these baselines. Major requirements changes to the system design or to performance that result in changes to external interfaces or that increase system resources are tracked to the baselines. These are the types of changes that are not made unilaterally by the development team, but are reviewed and approved by a higher authority. For operational systems that support the Flight Dynamics Facility, changes are normally tracked by the Code 550 Configuration Management Office (CMO) in support of the Flight Dynamics Facility Configuration Control Board (CCB).

There is a liaison between the configuration manager's duties as described here and the formal CCB procedures and the CMO. The configuration manager tracks changes resulting from the PDR and CDR and from major requirements changes and provides the necessary interface with the CMO.

Configuration management is a team effort. The configuration manager's responsibility is to keep track of all appropriate changes, not to identify the changes that need to be tracked. The latter responsibility falls to the technical manager, the task leader, the requirements team, and the PDR and CDR process. The procedures for conducting configuration management must therefore be highly visible to all development and requirements personnel at the beginning of the project and when new personnel join the project. Figure 4-1 depicts the configuration management process.

The purpose of this section is to present the policies and procedures for performing configuration management activities

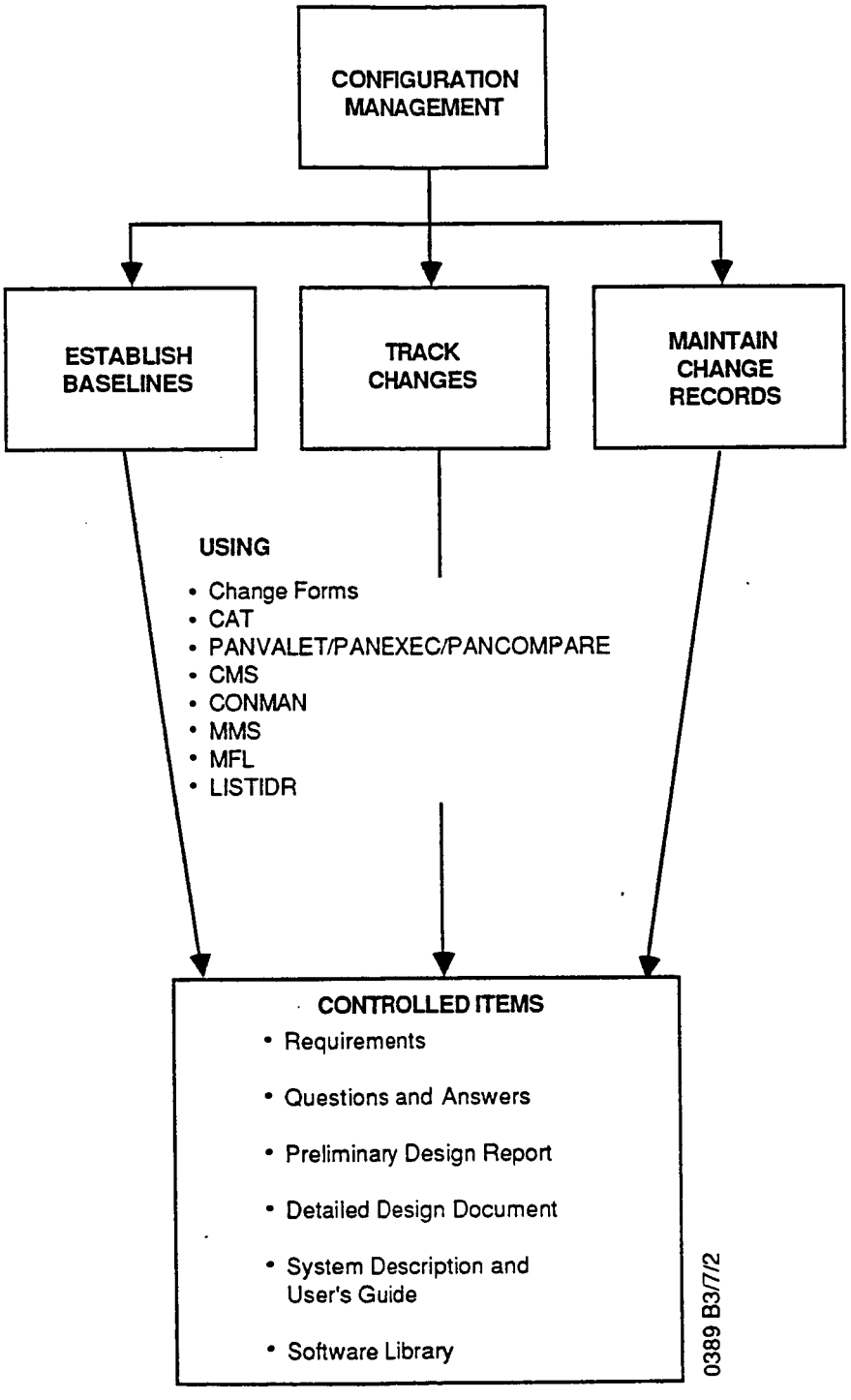

Figure 4-1. The Configuration Management Process

to support flight dynamics software development. It is organized as follows: First, the configuration management stage is set by presenting management considerations for planning and the configuration management plan. Second, configuration management methods are discussed along with the tools that are candidates to support the configuration management process. Third, the products that are normally configuration managed and the processes, that is, the applications of the methods and tools to these products, are described. Fourth, all of the previous information is summarized in one page of configuration management activities for each life-cycle phase of development.

4.1 MANAGEMENT CONSIDERATIONS

4.1.1 PLANNING

Planning for configuration management activities is one of the initial steps in the software development life cycle and is documented in the Configuration Management Plan as part of the Software Development/Management Plan. Early planning is necessary for configuration management procedures to be well established and integrated throughout all phases of the software development activities supporting the Flight Dynamics Facility.

The major planning decisions to be made before the Configuration Management Plan is developed involve identifying the products and product baselines that will be managed and the types of changes that will be tracked to the product baselines. Another key decision is the selection of the person (or persons) responsible for the configuration management process.

Planning considerations for the amount of effort required for configuration management activities are discussed in Section 2.2. The process described in this document is well established for attitude ground support system development efforts on IBM hardware. For this type of system, the actions

described for maintaining change records are necessary to support the Code 550 CMO and the Flight Dynamics Facility CCB and to maintain a controlled software library.

For other types of projects, such as experimental or research projects, the types of documents and document baselines described here may not be produced, and the CCB may not be involved. In those cases, the technical manager must identify the documents to be monitored and the level of effort to be expended managing changes to them. Decisions will be made based on the following considerations: Is the document a deliverable? How critical is it to the overall development effort? Are only changes of substance to be tracked, or are all changes to be tracked, including typographical error corrections and statements of clarification? The general guideline is to track changes in requirements that affect performance, changes in design that affect system resources, and any changes to external interfaces. Controlled software libraries are normally established as baselines for all project types, and changes are always tracked to the libraries.

The person selected as the configuration manager should be knowledgeable in the configuration management process and also have the tenaciousness to follow up as a matter of practice. Followup is essential in change management to ensure that every tracked change is incorporated into the appropriate product. The configuration manager is responsible for keeping the configuration management methods, tools, and milestones visible throughout the development cycle. Identifying a system librarian to maintain the controlled software library will be the minimum control point of any development effort, regardless of the size of the project. The librarian will maintain control of the software throughout the system development activity.

4.1.2 CONFIGURATION MANAGEMENT PLAN

The Configuration Management Plan specifies the procedures to be applied to the software and specific documents to ensure a controlled and systematic software development effort. The plan also assigns responsibility for configuration management. The technical manager is responsible for this plan and tailors it to the specific project. The plan addresses the following:

- Product baselines to be placed under configuration management control
- Changes to be tracked to each baseline
- Approval authority for each change
- Specific procedures to be followed by all team members for each product baseline
- Methodologies/tools to be used
- Identification of change forms to be used in relation to specific products
- Location of all change forms for central reference
- Identification of the configuration manager
- Procedures for dealing with external interfaces
  - Code 550 CMO/CCB
  - Requirements team

Appendix B presents guidelines for preparing a Configuration Management Plan.

4.2 CONFIGURATION MANAGEMENT METHODS AND TOOLS

4.2.1 CONFIGURATION MANAGEMENT METHODS

The primary methods used in the configuration management process are as follows:

- Establishing product baselines and specifying the types of changes to be tracked to each baseline
- Tracking changes and maintaining change records against the product baselines

Product baselines are established at specific points in the software development activity. Baselines represent a state of completeness or a culmination of activities, a formal departure point for controlling future changes. For example, the Preliminary Design Report is established as a baseline at the time of the PDR, when it represents the culmination of the preliminary design phase. Changes are tracked to this report until the Detailed Design Document baseline is established. The product baselines that are normally established for configuration management control are as follows:

- Preliminary Design Report
- Detailed Design Document
- User's Guide and System Description
- Controlled Software Library--Prologs, PDL, and code

The types of changes that are tracked for the product baselines are major requirements changes normally requiring formal CCB review and approval for operational systems supporting the Flight Dynamics Facility and all changes to controlled software libraries.

Change records are maintained to keep track of change activity. They are the responsibility of the configuration manager and are maintained in a central location in a configuration management notebook so that they are accessible to all development personnel.

4.2.2 CONFIGURATION MANAGEMENT TOOLS

The tools normally used in the configuration management process are as follows:

- Configuration Management Forms
- Configuration Analysis Tool (CAT)
- Controlled Software Library Tools--PANVALET/PANEXEC/PANCOMPARE, Code Management System (CMS), CONMAN
- Module Management System (MMS)
- Master File List (MFL)
- LISTIDR

4.2.2.1 Configuration Management Forms

A series of forms is used to track changes to flight dynamics software development activities. Appendix D contains the change forms that are described in this section. These forms are commonly used for configuration management control of operational development efforts such as attitude ground support systems. They are not intended to be all inclusive, and the technical manager must develop additional forms or tailor existing forms for a specific project.

Change forms are used to document questions resulting from PDRs and CDRs regarding requirements, design, and system performance. The Review Item Disposition (RID) is used to record the questions and the answers. The configuration manager is responsible for ensuring that the questions and answers are documented and forwarded to the CMO and that any resulting changes are made to the appropriate design document.

The Requirements/Specifications Question and Answer Form is used for questions between the system developers and the requirements team. These individuals complete the form, and the configuration manager ensures that the questions are answered. Any question and answer that results in a major

design/performance change is formally tracked for CMO/CCB action.

Requirements changes are documented on the Configuration Change Request (CCR), which is submitted to the Code 550 CMO and then passed to the CCB for disposition. The configuration manager records that a CCR has been filled out and tracks the changes that are approved to the appropriate product.

If the change is approved, a Document Change Notice (DCN) is prepared, and the configuration manager tracks and records the final disposition of the change in the appropriate document. The question-and-answer process goes on throughout the development activity.

A Component Origination Form (COF) is completed by the developers for each module that has been coded, code read, and unit tested before it is added to the controlled software library. These forms are used during the implementation phase and are submitted to the project librarian/configuration manager before modules are added to the controlled software libraries.

A Change Report Form (CRF) is also completed by the developers whenever a change is made to any module in the controlled software library or libraries. Changes are not made unless the change request is approved, normally by the technical manager, and submitted to the configuration manager, who makes the changes to the library.

4.2.2.2 Configuration Analysis Tool (CAT)

The Configuration Analysis Tool, available on both the VAX and IBM computers, is used to track and report project status. CAT is used to keep track of the following: discrepancies and changes, specification modifications, questions between the requirements team and the developers, and the RID forms. CAT provides management reports on each of these and is used throughout the development cycle. The configuration manager

normally provides the input for CAT, and the reports are used by the configuration manager as a followup tool and by the technical manager as a project status indicator. References 8 and 9 discuss CAT in more detail.

4.2.2.3 Controlled Software Library Tools

PANVALET, PANVALET/COMPARE, and PANEXEC are library management tools available on the IBM computer. PANVALET allows controlled access to source code libraries and maintains information about each member of the library (e.g., date of last access to a member, date of last maintenance of a member, and level numbers that indicate how many times the member has been changed and identify specific statements that were changed within the member). PANVALET/COMPARE provides a way of determining differences between two source code modules by performing software comparisons and generating reports on these comparisons. PANEXEC allows controlled access to object and execution module libraries and provides cross-referencing between source and execution modules. PANVALET is the primary tool of the configuration manager and is used to initiate the controlled library baseline at the beginning of the implementation phase. It is used until the system delivery file is delivered at the end of the project. Reference 10 discusses PANVALET in more detail.

The Code Management System (CMS) is a source library control system for the VAX computer. Its use is similar to PANVALET. References 11 and 12 describe CMS in more detail.

CONMAN is a library support tool available on the VAX and PDP computers that compares two libraries and produces a list of differences between them. It is used to compare the controlled version of files listed in a Master File List (see Section 3.2.2.6) with an updated version. It is also used to produce command procedure files that will move only the changed and new software components into a controlled area.

CONMAN is used during the implementation phase when updates are made to the controlled software library and in subsequent phases until the system delivery file is created and delivered to the customer.

4.2.2.4 Module Management System

MMS is a tool available on the VAX computer that can be used to track the lineage of system execution modules (see Section 3.2.2.5). MMS should be used to keep track of changes to execution modules when they are initially built from the controlled software library(s) during the implementation phase. MMS reports are kept by the configuration manager in a central location for reference by all team members.

4.2.2.5 Master File List

The MFL, as described in Section 3.2.2.6, is used to facilitate the tracking of differences between execution modules.

4.2.2.6 LISTIDR

LISTIDR is a program available on the IBM computer that provides an update history of execution modules. This program can be used to keep track of changes to the system's execution modules throughout the implementation phase and the subsequent test phases of development. LISTIDR is used when execution modules are first built from source code residing on the controlled software library or libraries. The individual creating the execution module runs LISTIDR, but the results are passed to the configuration manager to be maintained in a central location for reference.

4.3 CONFIGURATION MANAGEMENT PRODUCTS AND PROCESSES

The products that will normally be established for configuration management control are as follows:

- Documents
  - Preliminary Design Report
  - Detailed Design Document
  - User's Guide and System Description
- Software--Controlled software library, prologs, PDL, and code
- Other Items
  - Requirements/design decisions that remain to be determined
  - Questions and answers between the requirements team and the developers

4.3.1 DOCUMENTS

4.3.1.1 Preliminary Design Report

This document is established as a baseline at the time of the PDR during the preliminary design phase of the development effort. It remains under configuration control until the Detailed Design Document baseline is established. The specific procedure for change management of this document is as follows.

The configuration manager maintains the list of changes to be made to the document in response to comments from the PDR. The configuration manager is responsible for ensuring that appropriate RIDs are filled out for all items identified during the PDR and that they are submitted to the Code 550 CMO for disposition by the CCB. Approved changes are then documented with a DCN and produced as change pages to the document. The configuration manager is responsible for ensuring that change pages are distributed to the appropriate individuals. Through this process, the configuration manager keeps

track of each RID and each resulting change to ensure that all are accounted for.

4.3.1.2 Detailed Design Document

This document is established as a baseline at the time of the CDR during the detailed design phase of the development effort. It remains under configuration control until the System Description Document is established as the baseline. Changes as a result of the CDR are tracked as described above for the Preliminary Design Report.

4.3.1.3 User's Guide and System Description

These documents, or a single document that includes both, are established as baselines when they are delivered as preliminary documents in the system testing phase of development. Although these documents are normally drafted during the implementation phase, they are not placed under configuration control until they become deliverables as preliminary documents in the system testing phase. They remain under configuration control until they are delivered at the end of the project.

4.3.1.4 Tracking Requirements Changes to Documents Outside the PDR/CDR Process

Requirements changes or major changes to system design or performance requirements outside the PDR/CDR process are documented with a CCR. The form is filled out by the requirements team, the technical manager, or the task leader. It is submitted to the CMO by the configuration manager and tracked by the configuration manager until the disposition of the change, normally as change pages into the Preliminary Design Report, the Detailed Design Document, or the System Description baseline, depending on the phase of development.

4.3.2 SOFTWARE

Software libraries initiated during the preliminary design phase are used to establish the controlled baseline at the beginning of the implementation phase, normally using a library management system such as PANVALET for the IBM computer or CMS for the VAX computer. When established, the controlled baseline contains reusable modules.

Tracking changes to the controlled software library is the major activity of the configuration manager during the implementation phase of development as each newly coded module is added to the library. Each component is documented on a COF before it is added to the library. As each module is integration tested from execution load modules built from the controlled library, any errors detected during testing and their solution are documented on a CRF and approved by the task leader or the person designated in the Configuration Management Plan. When the change is approved, the configuration manager makes the changes to the library and maintains a copy of all CRFs. The configuration manager is the only person with write access to the controlled library, except for appropriate personnel (e.g., the task leader) for backup when the configuration manager is not available.

The configuration manager must also maintain sufficient backup files of the controlled software library, not only for insurance if the current library is destroyed, but also to be used when there is a problem in building execution modules from corrected code. If an error occurs that prohibits the creation of the execution module, the previous version of the library must be restored and then used to build execution modules for testing and subsequent updating.

Specific procedures will have to be established based on the specific project for generating object code and execution modules from the controlled library, for determining where error

corrections will be made and tested (e.g., on the user's library) before the controlled library is changed, for regression testing when needed, and for the controlled software library's file backup.

CONMAN, if used, will account for and track changes between succeeding controlled software libraries.

4.3.3 OTHER ITEMS

The configuration manager also tracks requirements or design decisions that remain to be determined and questions and answers between the requirements team and the developers. These items are part of the evolutionary development of a software development project. They generally clarify ambiguous requirements or design materials and sometimes result in major changes to the system.

The configuration manager uses CAT, if it is specified for use in the Configuration Management Plan, to list and record the questions and answers and progress on the requirements or design decisions that have not been determined. The configuration manager gathers the information for entry into CAT and distributes the CAT reports, and also follows up periodically on the decisions that remain and on any questions that are unanswered for more than 2 weeks. Questions and answers that result in major requirements changes, such as those affecting system performance, are processed as described in Section 4.3.1.4.

4.4 CONFIGURATION MANAGEMENT LIFE-CYCLE-PHASE SUMMARY

The configuration management actions that are taken in each phase of the software development life-cycle are summarized in the following subsections.

4.4.1 CONFIGURATION MANAGEMENT IN THE REQUIREMENTS ANALYSIS PHASE

(CM=Configuration Manager, TL=Task Leader, TM=Technical Manager)

Baselines Established

- None

Actions

- Develop Configuration Management Plan in Software Development/Management Plan (TM)
- Establish configuration management procedures (change folders, change forms, etc.) and publicize to all team members (CM)

4.4.2 CONFIGURATION MANAGEMENT IN THE PRELIMINARY DESIGN PHASE

(CM=Configuration Manager, TL=Task Leader, TM=Technical Manager)

Baselines Established

- Preliminary Design Report

Actions

- Track requirements changes
  - Document with CCR form (TM, TL, or member of requirements team)
  - Submit form to CMO (CM)
  - Track change if approved into Preliminary Design Report (CM)
- Track questions and answers between requirements team and developers
  - Submit all questions and answers through CM
  - Record question and date asked, and date answered when available, normally using CAT (CM)
  - Follow up on any question unanswered for more than 2 weeks (CM)
  - Track any resulting requirements changes from this process as described above
- Preliminary Design Report
  - Establish preliminary document used as basis for PDR as baseline (TM)
  - Monitor RID form completion and disposition as a result of PDR (CM)
  - Serve as interface with CMO (CM)
  - Track changes resulting from approved RIDs to final report (CM)

4.4.3 CONFIGURATION MANAGEMENT IN THE DETAILED DESIGN PHASE

(CM=Configuration Manager, TL=Task Leader, TM=Technical Manager)

Baselines Established

- Detailed Design Document

Actions

- Track requirements changes
  - Document with CCR form (TM, TL, or member of requirements team)
  - Submit form to CMO (CM)
  - Track change if approved into Detailed Design Document (CM)
- Track questions and answers between requirements team and developers
  - Submit all questions and answers through CM
  - Record question and date asked, and date answered when available, normally using CAT (CM)
  - Follow up on any question unanswered for more than 2 weeks (CM)
  - Track any resulting requirements changes from this process as described above
- Detailed Design Document
  - Establish preliminary document used as basis for CDR as baseline (TM)
  - Monitor RID form completion and disposition as a result of CDR (CM)
  - Serve as interface with CMO (CM)
  - Track changes resulting from approved RIDs to final report (CM)

4.4.4 CONFIGURATION MANAGEMENT IN THE IMPLEMENTATION PHASE

(CM=Configuration Manager, TL=Task Leader, TM=Technical Manager, DT=Development Team Member)

Baselines Established

- Controlled software library

Actions

- Track requirements changes
  - Document with CCR form (TM, TL, or member of requirements team)
  - Submit form to CMO (CM)
  - Track change if approved into Detailed Design Document (CM)
- Track questions and answers between requirements team and developers
  - Submit all questions and answers through CM
  - Record question and date asked, and date answered when available, normally using CAT (CM)
  - Follow up on any question unanswered for more than 2 weeks (CM)
  - Track any resulting requirements changes from this process as described above
- Establish controlled software library baseline, composed of reusable code identified in previous two design phases (CM)
- Track changes to controlled software library
  - Prepare COF or equivalent and submit to CM for each newly coded module after it has been unit tested for addition to library (DT)
  - Prepare CRFs for changes to coded modules to correct errors detected in integration testing and submit to CM (DT)
  - Update library with change indicated on CRF and retain copies of CRFs (CM)
  - Provide appropriate number of backup copies of controlled software library (CM)

4.4.5 CONFIGURATION MANAGEMENT IN THE SYSTEM TEST PHASE

(CM=Configuration Manager, TL=Task Leader, TM=Technical Manager, DT=Development Team Member)

Baselines Established

- User's Guide and System Description

Actions

- Track requirements changes
  - Document with a CCR form (TM, TL, or member of requirements team)
  - Submit form to CMO (CM)
  - Track change if approved into Detailed Design Document or User's Guide and System Description when baseline is established (CM)
- Track questions and answers between requirements team and developers
  - Submit all questions and answers through CM
  - Record question and date asked, and date answered when available, normally using CAT (CM)
  - Follow up on any question unanswered for more than 2 weeks (CM)
  - Track any resulting requirements changes from this process as described above
- Track changes to the controlled software library
  - Prepare COF or equivalent and submit to CM for each newly coded module after it has been unit tested for addition to library (DT)
  - Prepare CRFs for changes to coded modules to correct errors detected in system testing and submit to CM (DT)
  - Update library with change indicated on CRF and retain copies of CRF (CM)
  - Provide appropriate number of backup copies of controlled software library (CM)
- Establish User's Guide and System Description baseline (TM)

4.4.6 CONFIGURATION MANAGEMENT IN THE ACCEPTANCE TEST PHASE

(CM=Configuration Manager, TL=Task Leader, TM=Technical Manager, DT=Development Team Member)

Baselines Established

- None

Actions

- Track requirements changes
  - Document with a CCR form (TM, TL, or member of requirements team)
  - Submit form to CMO (CM)
  - Track change if approved into User's Guide and System Description
- Track questions and answers between requirements team and developers
  - Submit all questions and answers through CM
  - Record question and date asked, and date answered when available, normally using CAT (CM)
  - Follow up on any question unanswered for more than 2 weeks (CM)
  - Track any resulting requirements changes from this process as described above
- Track changes to controlled software library
  - Prepare COF or equivalent and submit to CM for each newly coded module after it has been unit tested for addition to library (DT)
  - Prepare CRFs for changes to coded modules to correct errors detected in acceptance testing and submit to CM (DT)
  - Update library with change indicated on CRF and retain copies of CRF (CM)
  - Provide appropriate number of backup copies of controlled software library (CM)

APPENDIX A - QUALITY ASSURANCE PLAN GUIDELINES

This appendix presents guidelines for preparing the Quality Assurance Plan, which is normally developed as part of the Software Development/Management Plan. The guidelines include quality assurance activities for each software product tailored for a typical attitude ground support system development effort. Quality Assurance Plans for other types of projects must be tailored to the specific project. The general guideline is to identify standards and a quality assurance process for each project deliverable to ensure that what is expected to be delivered is delivered (see Section 3.1).

1.0 INTRODUCTION

1.1 PURPOSE

Explicitly state why the plan is needed and how it is intended to be used by all development team members. The plan is not only for the quality assurer; it also contains the quality assurance activities that affect the entire system development effort. For example:

> The purpose of the Quality Assurance Plan is to establish the quality assurance procedures for this project by identifying the specific standards and guidelines to be used in the production of project documents and software. The plan is also to be used as a reference for all development team members for the standards and tools used during each phase of the development.

1.2 GOALS

Explicitly state what is to be achieved as a result of this plan, for example:

> The goal of this plan is to ensure that the quality assurance process is an integral part of the system development and that quality products are produced.

2.0 QUALITY ASSURANCE PROCEDURES

Quality assurance procedures are product oriented. They will be applied at the onset of each product development. The stage must be set for the development of each product by identifying the standards that are used as the guideline for the development and by ensuring that development team members understand the standards. After this is accomplished, the quality assurer must enforce the standards through appropriate spot checks and reviews and followup activity.

In the Quality Assurance Plan, the procedures that will be used for each project product will be described as follows.

2.1 QUALITY ASSURANCE PRODUCTS

- List the products that will be monitored for quality assurance.
- Discuss any major deviation from the list provided in Section 3.3 of this document.

2.1.1 Quality Assurance Procedures for Documents

For each document, discuss the following:

- **Schedule**
  - When does drafting of the document begin?
  - When is the document to be delivered, both preliminary and final, if applicable?
  - When will spot checks be made of the draft material for compliance with standards?
  - When will the document be reviewed for compliance with the identified standard?
  - Will a formal audit be made of this document? If so, when is it scheduled?
  - Is this document scheduled for review by the technical publications department? If so, when?

- Standards
  - What standard format will be used to develop the document?
  - Where is the standard referenced?
  - Are any deviations from the standard allowed?
  - What are the procedures for approval of deviations from the standard?
  - Who has approval authority for deviations from the standard?
- Checklists
  - What checklists will be used in preparing the document or in determining that the document is complete?
  - What are the procedures for deviations from the checklist, and who has approval authority for deviations?
- Process
  - Will quality assurance meetings be held to kick off the initiation of each new document?
  - What procedure will be followed for spot checking and reviewing the document?
  - Will there be one-on-one encounters between the quality assurer and the writer for spot checks, or will the writer be asked to submit the drafts to the quality assurer for independent review?
  - How will the formal reviews be conducted? Will the task leader and the technical manager be responsible for the final content of the document? Does the quality assurer play a role in

the review or does he/she only certify that the review has been made?

2.1.2 Quality Assurance Procedures for Software

For each software product, discuss the following:

- Schedule
  - When is the development of software components initiated? For example, when are program prologs and PDL first developed? When is coding begun? When is the system delivery file built?
  - When are the components to be completed? Describe by phase of development and builds within the implementation phase.
  - When are the quality assurance tools introduced for the software components?
  - When are spot checks, reviews, and audits of the software components to be made?
- Standards
  - What standards will be used to guide the development of the software components?
  - Where are the standards referenced?
  - Are any deviations from the standards allowed?
  - What are the procedures for approving deviations from the standards?
  - Who has approval authority for deviations from the standards?
- Tools
  - Will any automatic quality assurance tools be used? For example, will RXVP80 be used to analyze FORTRAN code as an adjunct to unit testing?

- Who will be responsible for making computer runs with the tools?
- What is the distribution of output from the tools?

- Process
  - How are spot checks made of the code? From developers' libraries before unit testing?
  - When is the code reviewed? Before submission to the controlled software library?
  - What is necessary for the review to be made? Source code listing and RXVP80 output?
  - What happens when the code does not meet the standard?
  - How is quality assurance approval achieved for each software component? What forms, what tools, who certifies, and who records?

2.2 EXTERNAL INTERFACES

- Will external organizations conduct formal audits? If so, they should be identified.

2.3 ROLE OF QUALITY ASSURER

Describe the specific duties of the quality assurer and how he/she will interface with the development team members. Discuss the following:

- Is the role of the quality assurer primarily as the tracker of quality assurance activities, or does he/she participate in the spot checks and reviews of the documents and the software?
- Does the quality assurer have the authority to direct developers who are falling short of the standards?

Or does the quality assurer report to the task leader or the technical manager to have the standards enforced?

- How does the quality assurer keep track of the quality assurance spot checks, reviews, and audits?
- When does the quality assurer know that enough spot checks have been made of either the document or software components to ensure standards compliance?

APPENDIX B - CONFIGURATION MANAGEMENT PLAN GUIDELINES

This appendix presents guidelines for preparing the Configuration Management Plan, which is normally developed as part of the Software Development/Management Plan. The guidelines include configuration management activities tailored for a typical attitude ground support system development effort. Configuration Management Plans for other types of projects must be tailored to the specific project. Particular attention is devoted to identifying the products to be placed under configuration management control, establishing the baselines for these products, and identifying the types of changes to be tracked to the product baselines (see Section 4.1).

1.0 INTRODUCTION

1.1 PURPOSE

Explicitly state why the plan is needed and how it is intended to be used by all development team members. The plan is not only for the configuration manager; it also contains the configuration management activities that affect the entire system development effort. In addition, it includes the activities required to interface with external organizations such as the requirements team and the Configuration Control Board (CCB). For example:

> The purpose of the Configuration Management Plan is to identify the products to be change management controlled, establish baselines for these products, define the types of changes to be tracked, and describe the configuration management procedures for each controlled product.

1.2 GOALS

Explicitly state what is to be achieved as a result of this plan. For example:

> The goal of this plan is to ensure that the configuration management process is fully integrated into the development effort and fully understood by all development personnel.

2.0 CONFIGURATION MANAGEMENT PROCEDURES AND TOOLS

Configuration management activities are software product oriented; the configuration management procedures and tools will be described as they relate to the products that are identified for configuration control.

2.1 CONTROLLED PRODUCTS

- List the products that will be controlled.
- Discuss any major deviations from the list provided in Section 4 of this document, for example: "No Preliminary Design Report will be produced." or "A Data Base Description Document will be produced."

2.1.1 Configuration Management Procedures for Documents

For each document discuss the following:

- Schedule
  - When is the baseline created? Include phase of development and calendar target date if possible, for example: "The Preliminary Design Report will be established as a baseline 5 days before the Preliminary Design Review."
  - How long does the baseline remain under control and what document replaces it, if applicable?

- Change
  - What kinds of changes will be tracked to the baseline? For example: "Major requirements changes that result in major design/performance/ resource changes will be tracked.
  - What forms will be used to track changes to the document baseline?
  - Who has the approval authority for the change?
  - Is this document under CCB control? If so, describe the interface procedures with the CCB.
  - Who is responsible for filling out the change form?
  - Who is responsible for making the change?
  - Where will the change forms be maintained?
  - Will informal changes be allowed? If so, what are they?
- Process
  - What is the procedure for establishing the baseline?
  - What does the baseline contain?
  - Where will the document baseline be kept, for example, as a PANVALET file or as a set of PC files, or both?
  - What kinds of changes will be made to the baseline and who will initiate them, who will approve them, and who will make them? What are the deviations from this process?

- How are the changes tracked after a change is initiated until it results in a document update? Are records kept of the changes after they have been made?

2.1.2 Configuration Management Procedures for Software Products

For software products, discuss the following:

- Schedule
  - When is the controlled software library baseline established?
- Baseline
  - Since software libraries are established at the beginning of the design phase of development, discuss the continuum of events from the design phase to the implementation phase when the controlled baseline is established as a controlled software library and the process for establishing the baseline. Identify the prefixes for major subsystems, utilities, and the naming conventions.
  - What are the contents of the baseline by software type?
  - What library management tools will be used to maintain the controlled software library(s)?
- Change
  - Where does the code to be modified reside?
  - What change forms are required?
  - Who reviews and/or approves the changes?

- Where are tests of pending changes run before they are added to the controlled library? (Are they run from the user's library?)
- Who has read and/or write access to the controlled library?
- What is the procedure for failure recovery?
- Who creates execution modules for testing?
- How often will the controlled library be updated? Is it updated for every change, or is it updated when a batch of changes are available? How are emergency changes handled?
- What is the criteria for adding new items to the library? Is it the same for all types of items?
- Is system growth to be monitored? If so, what procedures will be followed to track growth?
- Who approves the addition of new items to the library?
- What forms are used to add new items to the library?
- When is regression testing necessary?

• Process

- How is the controlled software library baseline established?
- What steps do the developers take to add or make changes to the controlled library? Will the code be code read and unit tested and approved before it is added to the library?

- How is the controlled library maintained from one system build to another? Is a separate library established for each system build, or is a single library used throughout the development effort?
- What are the update procedures during integration testing (build or release testing), system, and acceptance testing? Are there any differences that need to be identified?

• What are the backup and recovery procedures for the controlled software library?

2.2 EXTERNAL INTERFACES

• Requirements Team

- Are Question and Answer Forms used to track questions and answers between the requirements team and the developers?
- Who tracks the answers?
- Who determines the effect of the answers?
- What are the procedures for changes that result from the question-and-answer process?

• Configuration Management Office (CMO)/Configuration Control Board (CCB)

- Who is the point of contact for CMO/CCB transactions?
- What are the internal procedures for tracking changes that are submitted to the CCB?
- Who are the Review Item Disposition Forms sent to and received from?

- What are the procedures for handling RID responses?
  - Who prepares the Document Change Notices?
- Configuration Analysis Tool
  - Who inputs the data?
  - What is the frequency for each report?
  - What is the distribution of the CAT reports?

2.3 ROLE OF THE CONFIGURATION MANAGER

Describe the specific duties of the configuration manager and how he/she will interface with the development team members.

APPENDIX C - QUALITY ASSURANCE CHECKLISTS

The checklists presented in this appendix contain the information that will be covered for deliverable software development products that have been described in this document. The checklists are to be used as guidelines. For projects that do not follow the normal life-cycle development or do not produce the standard products, appropriate checklists will be developed and included in the Quality Assurance Plan. The following checklists are presented:

- Software Development/Management Plan Quality Assurance Checklist
- Requirements Analysis Summary Report Quality Assurance Checklist
- Preliminary Design Report Quality Assurance Checklist
- Detailed Design Document Quality Assurance Checklist
- PDL Standards Review Quality Assurance Checklist
- FORTRAN Code Quality Assurance Checklist
- Code Reading Quality Assurance Checklist
- Build/Release/System Test Plan Quality Assurance Checklist
- System Description Quality Assurance Checklist
- User's Guide Quality Assurance Checklist
- System Delivery File Quality Assurance Checklist

SOFTWARE DEVELOPMENT/MANAGEMENT PLAN QUALITY ASSURANCE CHECKLIST

1. Purpose of project
2. Relationship of project's software products to overall system
3. Development team size, staffing pattern, and team composition and organization
4. External group interfaces, points of contact, and group responsibilities
5. Development approach (technical and management) for every phase as follows:
   - Assumptions and constraints
   - Anticipated and unresolved problems
   - Major activities for the phase
   - List of development methods and tools
   - Products for the phase
   - Resource estimates (staff, computer, software reuse, documentation, software rehosting, and maintenance)
   - Definition of progress accounting in terms of metrics, data collection methods and forms, and summary report forms
6. Quality Assurance Plan
7. Configuration Management Plan
8. Major reviews (SRR, PDR, CDR)--Staff roles, schedule, attendees, materials
9. Milestone charts
   - Development life-cycle phases
   - Delivery of required external interfaces
   - Schedule dates for integration of externally developed software (if applicable)
   - Delivery of all development products
   - Internal reviews
   - Internal products

REQUIREMENTS ANALYSIS SUMMARY REPORT QUALITY ASSURANCE CHECKLIST

1. Purpose of project
2. Overall system concepts
3. System description--Baseline diagrams, hardware interfaces, external interfaces, data flow diagrams, operating scenarios with interfaces and data flow
4. System constraints
   - Hardware
   - Operating system limitations
   - Support software limitations
5. Areas of concern and TBD requirements
   - Assessment of their effects on system size, required effort, cost, and schedules
   - Priority assignment
6. System and subsystem requirements
   - Level of importance
   - Completeness
   - Input (frequency, volume, coordinates, units, and timing)
   - Output (frequency, volume, coordinates, units, and timing)
7. Data interfaces
   - Name, function, frequency, coordinates, units, data type
   - Organization (e.g., sequential), medium, record layouts, storage requirements
8. Summary of reusable code
9. Estimates--System size, required effort, cost, schedule

PRELIMINARY DESIGN REPORT QUALITY ASSURANCE CHECKLIST

1. Purpose of project
2. Overall system concepts
3. System description--Baseline diagrams, hardware interfaces, external interfaces, data flow diagrams, operating scenarios with interfaces and data flow
4. System constraints and effects on design
   - Hardware
   - Operating system limitations
   - Support software limitations
5. Areas of concern and TBD requirements
   - Assessment of their effects on system size, required effort, cost, and schedules
   - Priority assignment
   - Effects of assumptions on the design
6. Critique of alternative designs
7. Subsystem design descriptions
   - Tree diagrams expanded to two levels below the subsystem driver
   - Data flow diagrams for each processing mode
   - Processing related to operator-specified input and action: points of control, functions performed, normal results, abnormal results, error processing and recovery
8. System and subsystem resource requirements--Hardware, peripheral space, memory
9. Data interfaces
   - Name, function, frequency, coordinates, units, data type
   - Organization (e.g., sequential), medium, record layouts, storage requirements
10. Summary of reusable code

DETAILED DESIGN DOCUMENT QUALITY ASSURANCE CHECKLIST

1. Purpose of project
2. Overall system concepts
3. System description--Baseline diagrams, hardware interfaces, external interfaces, data flow diagrams, operating scenarios with interfaces and data flow
4. System constraints and effects on design
   - Hardware
   - Operating system limitations
   - Support software limitations
5. Areas of concern and TBD requirements
   - Assessment of their effects on system size, required effort, cost, and schedules
   - Priority assignment
   - Effects of assumptions on the design
6. Subsystem design description
   - Overall subsystem capability
   - Tree diagrams expanded to component level showing interfaces and control flow
   - Restrictions for each processing mode
   - Internal storage requirements for each processing mode
   - Detailed input and output specifications
   - List of all error messages including system response and operator action
   - Prologs and PDL for every subroutine
   - Processing related to operator-specified input and action: points of control, functions performed, normal results, abnormal results, error processing and recovery
7. System and subsystems resource requirements--Hardware, peripheral space, memory
8. Data interfaces
   - Name, function, frequency, coordinates, units, data type
   - Organization (e.g., sequential), medium, record layouts, storage requirements
9. Summary of reusable code

PDL STANDARDS REVIEW QUALITY ASSURANCE CHECKLIST

1. Use of subjects, verbs, and objects in all PDL statements
2. Use of only standard PDL control structure keywords, relational attributes, and prepositions
3. Special character precedes and follows all comments
4. PDL contains all variables and external references defined in the prolog
5. PDL nouns defined in the data dictionary
6. PDL contains all constraints, restrictions, abnormal terminations, messages, etc., identified in the prolog
7. Standard formats used for syntax for sequential, selection control, and iteration statements
8. Straightforward logic
9. Compound conditions standard and used only when necessary
10. Embedded PDL starts with a page break
11. PDL length of 1 page (60 lines) or less
12. Each PDL statement begins with a standard special character

FORTRAN CODE QUALITY ASSURANCE CHECKLIST

1. Use of parameters for
   - Computer-related characteristics (e.g., word and character sizes)
   - Design-dependent data attributes (e.g., table size and message length)
   - I/O device-oriented characteristics
   - Constants used to set flags and to reference data in lists and tables
3. Variable names describe the function of the variable
4. Data and character value definitions clear
5. Data appropriately initialized
6. Appropriate guidelines followed for
   - Module interfaces
   - Comments
   - Control processing
   - Loop processing
   - Code indentation
   - Statement format and labels
   - Computational processing
   - I/O processing
   - Hardware-dependent processing
7. Common file data logically related

CODE READING QUALITY ASSURANCE CHECKLIST

1. All constants defined
2. All unique values explicitly tested for input parameters
3. Values stored after they are calculated
4. Path defined for every possible outcome of a logical decision
5. All input defaults explicitly tested
6. All unique values of a keyword checked
7. Specified precision sufficient for required accuracy
8. Flagged/missing data values correctly excluded from computations
9. Display formats large enough
10. Data sets properly opened and closed
11. Leading and trailing records recognized and handled appropriately
12. Unit numbers unique
13. Double-precision constants used in double-precision expressions
14. Correct units or the appropriate conversion used (e.g., degrees, radians)
15. Correct sign (positive or negative) used, especially in bias adjustments
16. All counters properly initialized (0 or 1)
17. Absolute and symbolic (parameter) values used appropriately
18. On comparison of two bytes, all bits are compared, if necessary
19. Variable names informative
20. Prolog defines all variables used
21. Code indented to show its structure
22. Code adequately commented
23. Prolog defines operational constraints and assumptions
24. Continuation lines clearly indicated
25. Statement labels occur in an obvious sequence

BUILD/RELEASE/SYSTEM TEST PLAN QUALITY ASSURANCE CHECKLIST

1. Purpose of test plan
2. Type and level of testing
3. Testing schedule
4. Test description for every test
   - Specific capability or requirement tested
   - Detailed specification of input
   - Required environment (e.g., data sets and hardware)
   - Operational procedure
   - Detailed specification of output
   - Test result acceptance criteria

SYSTEM DESCRIPTION QUALITY ASSURANCE CHECKLIST

1. Purpose of project
2. Overall system concepts
3. System description--Baseline diagrams, hardware interfaces, external interfaces, data flow diagrams, operating scenarios with interfaces and data flow
4. Subsystem description
   - Overall capability
   - Assumptions and restrictions to processing
   - High-level subsystem data flow diagrams
   - Input and output
   - Tree diagrams to component level
5. System initiation procedures
   - Resource requirements with high level diagrams and tables for the system and subsystem for hardware, support data sets, peripheral space, memory, CPU time, I/O time
   - Specific initiation information
   - Program structure information (e.g., overlaying or loading control)
6. Detailed description of input and output for each step (source code libraries, object code libraries, execution code libraries, etc.)
7. Internal storage requirements--Description and size of arrays
8. Data interfaces
   - Name, function, frequency, coordinates, units, data type
   - Organization, medium, record layouts, storage requirements
9. Prologs and PDL
10. List and description of components from support data sets

USER'S GUIDE QUALITY ASSURANCE CHECKLIST

1. Purpose of project
2. Overall system concepts
3. System description--Baseline diagrams, hardware interfaces, external interfaces, data flow diagrams, operating scenarios with interfaces and data flow
4. Subsystem description
   - Overall capability
   - Assumptions and restrictions to processing
   - High-level subsystem data flow diagrams
   - Input and output
   - Description of subsystem input and output
   - Processing related to operator-specified input and action: points of control, functions performed, normal results, abnormal results, error processing and recovery
5. Execution requirements--Hardware, peripheral space, memory, control information for each processing mode, timing considerations (CPU time, I/O time, wall-clock time)
6. System and subsystem input and output description
   - Facsimiles of graphic displays in the order they are produced
   - Facsimiles of hardcopy output in the order produced and annotated to show controlling parameters
   - List of numbered error messages with system response and operator action

SYSTEM DELIVERY FILE QUALITY ASSURANCE CHECKLIST

1. First file (description and installation guide) contains
   - System name
   - Abstract of system function
   - List of all files on the tape with file number and name and description of file contents for every file
   - Operating environment (hardware and software requirements)
   - System unloading procedure
   - Supporting documentation references
2. All files to generate an execution load module include
   - Primary source code
   - Command procedures (e.g., JCL) to compile and link-edit the source code
   - Subroutine libraries
3. Every file to execute the system contains
   - Load module library
   - Support libraries
   - Command procedures (e.g., JCL) to execute the system
   - Permanent input and output data sets, including required initialization routines
4. Every file to test the system contains
   - Command procedures (e.g., JCL) to execute the system for each test
   - Permanent input data sets for each test
   - Output data for each test

APPENDIX D - STANDARD CONFIGURATION MANAGEMENT CHANGE FORMS

The forms presented in this appendix are used to record changes during the normal development process as described in this document. If new and different forms are necessary for a specific project, they will be included in the Configuration Management Plan. The following forms are presented:

- Review Item Disposition (RID)
- RID Response
- Requirements/Specifications Question and Answer Form
- Configuration Change Request (CCR)
- Document Change Notice (DCN)
- Component Origination Form (COF)
- Change Report Form (CRF)

FLIGHT DYNAMICS DIVISION	REVIEW ITEM DISPOSITION (RID)	RID NO
		DATE RECEIVED
		RECEIVED BY
DOCUMENT TITLE		REVIEW DATE
		SUBMISSION DATE

PROBLEM DESCRIPTION

RECOMMENDATION

IMPACT IF RECOMMENDATION NOT IMPLEMENTED

ORIGINATOR	ORGANIZATION	TELEPHONE

DISPOSITION: ☐ APPROVED BY ORIGINATOR ☐ DEFERRED TO CCB ☐ WAIVER REQUESTED

SPONSORED BY:

NASA MANAGER DATE

FDD/CM0(CSC)/011 (2/86)

Review Item Disposition (RID)

FLIGHT DYNAMICS DIVISION	RID RESPONSE	RESPONSE NO.
		DATE RECEIVED
		RECEIVED BY
DOCUMENT TITLE		RID NO.
		RESPONSE DATE

RESPONDENT'S RECOMMENDATIONS

REQUIRED DOCUMENT CHANGES (ATTACHED)

SECTION	PAGE	SECTION	PAGE	SECTION	PAGE

RESPONDENT	ORGANIZATION	TELEPHONE

RESPONSE APPROVAL: ☐ APPROVED ☐ REJECTED

ORIGINATOR DATE

NASA MANAGER DATE

FDD/CM0(CSC)/012 (2/86)

RID Response

REQUIREMENTS/SPECIFICATIONS QUESTION AND ANSWER FORM

NUMBER : _____

Date Submitted : _____	Date Answered : _____
Submitted By : _____	Answered By : _____
Date Answer Needed : _____	Resulting CCR # : _____
Affected Subsystems : _____	Approved By : _____

QUESTION/COMMENT/RECOMMENDATION :

Continuation Attached ☐

RESPONSE :

Continuation Attached ☐

Requirements/Specifications Question and Answer Form
(1 of 2)

REQUIREMENTS/SPECIFICATIONS QUESTION

NUMBER : _____

CONTINUATION

QUESTION/COMMENT/RECOMMENDATION :

Requirements/Specifications Question and Answer Form
(2 of 2)

FLIGHT DYNAMICS DIVISION CONFIGURATION CHANGE REQUEST (CCR)

1. CCR NO.	2. CONFIGURATION ITEM NAME	3. PRIORITY	4. CHANGE LEVEL
_____		☐ EMERGENCY ☐ URGENT ☐ ROUTINE	☐ 1 ☐ 3 ☐ 2 ☐ 4

5. TITLE OF CHANGE

6.
DOCUMENT TITLE

DOCUMENT NO.

7. REASON FOR CHANGE

8. DESCRIPTION OF CHANGE

9. IMPACT

☐ SCHEDULE	☐ RELIABILITY	☐ GROUND SEGMENT	☐ 551
☐ BUDGET	☐ MAINTAINABILITY	☐ SPACE SEGMENT	☐ 552
☐ FACILITIES	☐ USER SERVICES	☐ LOGISTICS	☐ 553
☐ TESTING	☐ SECURITY	☐ DOCUMENTATION	☐ 554
☐ TRAINING	☐ USAF FUNDING REQD	☐ HARDWARE	☐ OTHER
☐ CONTRACTOR SUPPORT	☐ POWER	☐ SOFTWARE	_____
☐ INTERFACES	☐ WEIGHT	☐ OTHER	_____

10. COMMENTS

11. ORIGINATOR	ORGANIZATION	TELEPHONE

12. AUTHORIZING SIGNATURE

_____

BRANCH HEAD DATE

FDD/CM0(CSC)/026 (4/86)

Configuration Change Request (CCR)

FLIGHT DYNAMICS DIVISION	**DOCUMENT CHANGE NOTICE (DCN)**	DCN NUMBER
		DATE
		CHANGE REQUEST REFERENCE
ORIGINATOR NAME	ORGANIZATION	TELEPHONE
DOCUMENT TITLE		DOCUMENT NUMBER
PAGES CHANGED (INCLUDING DELETIONS)		
		CONTINUED ☐
CMO SIGNATURE		DATE

CSC/OSSO/CMO/019(5/85)

Document Change Notice (DCN)

0389

COMPONENT ORIGINATION FORM

Component _____ Programmer _____

Subsystem _____ Date _____

Project _____

Location of source file

Library or directory _____ Member name _____

Relative difficulty of component

Please indicate (your judgement) by marking an X on the line below:

_____

Easy Medium Hard

Origin

If the component was modified or derived from a different project, please indicate the approximate amount of change and from where it was acquired; if it was coded new (from detailed design) indicate NEW.

_____ NEW

_____ Extensively modified (more than 25% of statements changed)

_____ Slightly modified

_____ Old (Unchanged)

If not new, where is it from ? _____

Type of Component

_____ 'INCLUDE' file (e.g., COMMON)

_____ JCL (or other control)

_____ ALC (assembler code)

_____ FORTRAN executable source

_____ Pascal source

_____ Ada source

_____ Namelists or parameter lists

_____ Display identification (GESS)

_____ Menu definition or help

_____ Reference data files

_____ BLOCK DATA file

_____ other (describe) _____

Purpose of Executable Component

For executable code, please identify the major purpose or purposes of this component. (Check all that apply).

_____ I/O processing

_____ Algorithmic/computational

_____ Data transfer

_____ Logic/decision

_____ Driver module

_____ Interface to operating system

GSFC 550-1 (3/85)

Component Origination Form (COF) (1 of 2)

Instructions for completing the Component Origination Form

This form is completed by the programmer for each source code component, when it is ready to be added to the controlled library (that is, placed under configuration control). There will be one CO form for each distinct component in the library. (For changes to components in the controlled library, use Change Report Forms.)

Header Information: self-explanatory. If you do not know the names used for project and subsystem, see the task leader or the configuration manager. The date is the date that the form is completed—that is, after all the coding is complete.

Location of Source: provide enough information for the librarian to move the file from your library into the controlled project library. This includes the device, the directory (on DEC machines) or library (on IBM), and the member or file name.

Difficulty: use your own judgement. The continuum is provided that you may distinguish among several components that you develop.

Origin: if the component is new (not copies from an earlier component) mark NEW. If you used pre-existing code, indicate how much you changed the old code and where you borrowed it from.

Type of Component: select the one description that best typifies this component. HELP files are classed with MENU files. Reference data includes constants or parameters that are not subject to modification (format definitions . . .).

Purpose of Executable Component: select one or more functions that describe the major purpose(s) of this component.

Component Origination Form (COF) (2 of 2)

0389

CHANGE REPORT FORM

PROJECT NAME _____ CURRENT DATE _____

PROGRAMMER NAME _____ APPROVED BY _____

SECTION A – IDENTIFICATION

DESCRIBE THE CHANGE: (What, why, how) _____

EFFECT: What components (or documents) are changed? (Include version) _____

EFFORT: What additional components (or documents) were examined in determining what change was needed? _____ _____

	(Month)	Day	Year)
Need for change determined on			
Change completed (incorporated into system)			

	1hr/less	1hr/1dy	1dy/3dys	>3dys
Effort in person time to isolate the change (or error)				
Effort in person time to implement the change (or correction)				

SECTION B – ALL CHANGES

TYPE OF CHANGE (Check one)

- ☐ Error correction
- ☐ Planned enhancement
- ☐ Implementation of requirements change
- ☐ Improvement of clarity, maintainability, or documentation
- ☐ Improvement of user services
- ☐ Insertion/deletion of debug code
- ☐ Optimization of time/space/accuracy
- ☐ Adaptation to environment change
- ☐ Other (Explain on back)

EFFECTS OF CHANGE

Y	N	
☐	☐	Was the change or correction to one and only one component?
☐	☐	Did you look at any other component?
☐	☐	Did you have to be aware of parameters passed explicitly or implicitly (e.g., common blocks) to or from the changed component?

SECTION C – FOR ERROR CORRECTIONS ONLY

SOURCE OF ERROR (Check one)

- ☐ Requirements
- ☐ Functional specifications
- ☐ Design
- ☐ Code
- ☐ Previous change

CLASS OF ERROR (Check most applicable)*

- ☐ Initialization
- ☐ Logic/control structure (e.g., flow of control incorrect)
- ☐ Interface (internal) (module to module communication)
- ☐ Interface (external) (module to external communication)
- ☐ Data (value or structure) (e.g., wrong variable used)
- ☐ Computational (e.g., error in math expression)

*If two are equally applicable, check the one higher on the list.

CHARACTERISTICS (Check Y or N for all)

Y	N	
☐	☐	Omission error (e.g., something was left out)
☐	☐	Commission error (e.g., something incorrect was included)
☐	☐	Error was created by transcription (clerical)

FOR LIBRARIANS USE ONLY

NUMBER _____

DATE _____

BY _____

CHECKED BY _____

	(Month	Day	Year)
ORIGIN DATE			

(Additional Comments on Reverse Side)

Change Report Form (CRF)

REFERENCES

1. Software Engineering Laboratory, SEL-81-205, *Recommended Approach to Software Development*, F. E. McGarry, G. Page, et al., April 1983

2. --, SEL-84-001, *Manager's Handbook for Software Development*, W. W. Agresti, F. E. McGarry, et al., April 1984

3. --, SEL-86-01, *Programmer's Handbook for Flight Dynamics Software Development*, R. Wood and E. Edwards, March 1986

4. --, SEL-85-005, *Software Verification and Testing*, D. Card, E. Edwards, et al., December 1985

5. *RXVP80, The Verification and Validation System for FORTRAN 4.0 User's Manual*, General Research Corporation, 1983

6. Software Engineering Laboratory, SEL-78-302, *FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3)*, July 1986

7. Digital Equipment Corporation, *VAX DEC/MMS User's Guide*, August 1984

8. Software Engineering Laboratory, SEL-81-007, *Software Engineering Laboratory (SEL) Compendium of Tools*, W. J. Decker et al., February 1981

9. --, SEL-80-104, *Configuration Analysis Tool (CAT) System Description and User's Guide (Revision 1)*, W. J. Decker and W. Taylor, December 1982

10. Computer Sciences Corporation, CSC/SD-86/6031, *FD/SDE User's Guide*, J. Buell and W. Miller, December 1986

11. Digital Equipment Corporation, *User's Introduction to VAX DEC/CMS*, November 1984

12. Digital Equipment Corporation, *VAX DEC/CMS Reference Manual*, November 1984

0389

C-2

STANDARD BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, Proceedings From the First Summer Software Engineering Workshop, August 1976

SEL-77-002, Proceedings From the Second Summer Software Engineering Workshop, September 1977

SEL-77-004, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977

SEL-77-005, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-005, Proceedings From the Third Summer Software Engineering Workshop, September 1978

SEL-78-006, GSFC Software Engineering Research Requirements Analysis Study, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, Applicability of the Rayleigh Curve to the SEL Environment, T. E. Mapp, December 1978

SEL-78-302, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3), W. J. Decker and W. A. Taylor, July 1986

SEL-79-002, The Software Engineering Laboratory: Relationship Equations, K. Freburger and V. R. Basili, May 1979

SEL-79-003, Common Software Module Repository (CSMR) System Description and User's Guide, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 1979

SEL-79-004, Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-003, Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study, T. Welden, M. McClellan, and P. Liebertz, May 1980

SEL-80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook and F. E. McGarry, December 1980

SEL-81-008, Cost and Reliability Estimation Models (CAREM) User's Guide, J. F. Cook and E. Edwards, February 1981

SEL-81-009, Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation, W. J. Decker and F. E. McGarry, March 1981

SEL-81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

SEL-81-012, The Rayleigh Curve As a Model for Effort Distribution Over the Life of Medium Scale Software Systems, G. O. Picasso, December 1981

SEL-81-013, Proceedings From the Sixth Annual Software Engineering Workshop, December 1981

SEL-81-014, Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-81-101, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

SEL-81-102, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide Revision 1, P. Lo and D. Wyckoff, July 1983

SEL-81-104, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

SEL-81-106, Software Engineering Laboratory (SEL) Document Library (DOCLIB) System Description and User's Guide, W. Taylor and W. J. Decker, May 1985

SEL-81-107, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, W. A. Taylor, and E. J. Smith, February 1982

SEL-81-110, Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics, G. Page, F. E. McGarry, and D. N. Card, June 1985

SEL-81-203, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, P. Lo, June 1984

SEL-81-205, Recommended Approach to Software Development, F. E. McGarry, G. Page, S. Eslinger, et al., April 1983

SEL-82-001, Evaluation of Management Measures of Software Development, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-003, Software Engineering Laboratory (SEL) Data Base Reporting Software User's Guide and System Description, P. Lo, August 1983

SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982

SEL-82-007, Proceedings From the Seventh Annual Software Engineering Workshop, December 1982

SEL-82-008, Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory, V. R. Basili and D. M. Weiss, December 1982

SEL-82-102, FORTRAN Static Source Code Analyzer Program (SAP) System Description (Revision 1), W. A. Taylor and W. J. Decker, April 1985

SEL-82-105, Glossary of Software Engineering Laboratory Terms, T. A. Babst, F. E. McGarry, and M. G. Rohleder, October 1983

SEL-82-406, Annotated Bibliography of Software Engineering Laboratory Literature, D. N. Card, Q. L. Jordan, and F. E. McGarry, November 1986

SEL-83-001, An Approach to Software Cost Estimation, F. E. McGarry, G. Page, D. N. Card, et al., February 1984

SEL-83-002, Measures and Metrics for Software Development, D. N. Card, F. E. McGarry, G. Page, et al., March 1984

SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983

SEL-83-006, Monitoring Software Development Through Dynamic Variables, C. W. Doerflinger, November 1983

SEL-83-007, Proceedings From the Eighth Annual Software Engineering Workshop, November 1983

SEL-84-001, Manager's Handbook for Software Development, W. W. Agresti, F. E. McGarry, D. N. Card, et al., April 1984

SEL-84-002, Configuration Management and Control: Policies and Procedures, Q. L. Jordan and E. Edwards, December 1984

SEL-84-003, Investigation of Specification Measures for the Software Engineering Laboratory (SEL), W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

SEL-84-004, Proceedings From the Ninth Annual Software Engineering Workshop, November 1984

SEL-85-001, A Comparison of Software Verification Techniques, D. N. Card, R. W. Selby, Jr., F. E. McGarry, et al., April 1985

SEL-85-002, Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team, R. Murphy and M. Stark, October 1985

SEL-85-003, Collected Software Engineering Papers: Volume III, November 1985

SEL-85-004, Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics, R. W. Selby, Jr., May 1985

SEL-85-005, Software Verification and Testing, D. N. Card, C. Antle, and E. Edwards, December 1985

SEL-85-006, Proceedings From the Tenth Annual Software Engineering Workshop, December 1985

SEL-86-001, Programmer's Handbook for Flight Dynamics Software Development, R. Wood and E. Edwards, March 1986

SEL-86-002, General Object-Oriented Software Development, E. Seidewitz and M. Stark, August 1986

SEL-86-003, Flight Dynamics System Software Development Environment Tutorial, J. Buell and P. Myers, July 1986

SEL-86-004, Collected Software Engineering Papers: Volume IV, November 1986

SEL-86-005, Measuring Software Design, D. N. Card, October 1986

SEL-86-006, Proceedings From the Eleventh Annual Software Engineering Workshop, December 1986

SEL-87-001, Product Assurance Policies and Procedures for Flight Dynamics Software Development, S. Perry et al., March 1987

SEL-RELATED LITERATURE

Agresti, W. W., Definition of Specification Measures for the Software Engineering Laboratory, Computer Sciences Corporation, CSC/TM-84/6085, June 1984

[4]Agresti, W. W., V. E. Church, D. N. Card, and P. L. Lo, "Designing With Ada for Satellite Simulation: A Case Study," Proceedings of the First International Symposium on Ada for the NASA Space Station, June 1986

[2]Agresti, W. W., F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," Program Transformation and Programming Environments. New York: Springer-Verlag, 1984

[1]Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1981

[1]Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: IEEE Computer Society Press, 1980 (also designated SEL-80-008)

[3]Basili, V. R., "Quantitative Evaluation of Software Methodology," Proceedings of the First Pan-Pacific Computer Conference, September 1985

[1]Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?", Journal of Systems and Software, February 1981, vol. 2, no. 1

[1]Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

[3]Basili, V. R., and N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," Proceedings of the International Computer Software and Applications Conference, October 1985

[4]Basili, V. R., and D. Patnaik, A Study on Fault Prediction and Reliability Assessment in the SEL Environment, University of Maryland, Technical Report TR-1699, August 1986

[2]Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," Communications of the ACM, January 1984, vol. 27, no. 1

[1]Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

[3]Basili, V. R., and C. L. Ramsey, "ARROWSMITH-P--A Prototype Expert System for Software Engineering Management," Proceedings of the IEEE/MITRE Expert Systems in Government Symposium, October 1985

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity, and Cost. New York: IEEE Computer Society Press, 1979

[2]Basili, V. R., R. W. Selby, and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," IEEE Transactions on Software Engineering, November 1983

[3]Basili, V. R., and R. W. Selby, Jr., "Calculation and Use of an Environments's Characteristic Software Metric Set," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

Basili, V. R., and R. W. Selby, Jr., Comparing the Effectiveness of Software Testing Strategies, University of Maryland, Technical Report TR-1501, May 1985

[4]Basili, V. R., R. W. Selby, Jr., and D. H. Hutchens, "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, July 1986

[2]Basili, V.R., and D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data, University of Maryland, Technical Report TR-1235, December 1982

[3]Basili, V. R., and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, November 1984

[1]Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

[1]Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

[1]Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: IEEE Computer Society Press, 1978

[3]Card, D. N., "A Software Technology Evaluation Program," Annais do XVIII Congresso Nacional de Informatica, October 1985

[4]Card, D. N., V. E. Church, and W. W. Agresti, "An Empirical Study of Software Design Practices," IEEE Transactions on Software Engineering, February 1986

[3]Card, D. N., G. T. Page, and F. E. McGarry, "Criteria for Software Modularization," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

[1]Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceed- ings of the Fifth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1981

[4]Church, V. E., D. N. Card, W. W. Agresti, and Q. L. Jordan, "An Approach for Assessing Software Proto- types," ACM Software Engineering Notes, July 1986

[2]Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," Proceedings of the Seventh International Computer Software and Applications Conference. New York: IEEE Computer Society Press, 1983

Higher Order Software, Inc., TR-9, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977 (also designated SEL-77-005)

[3]McGarry, F. E., J. Valett, and D. Hall, "Measuring the Impact of Computer Resource Quality on the Software Develop- ment Process and Product," Proceedings of the Hawaiian Inter- national Conference on System Sciences, January 1985

[3]Page, G., F. E. McGarry, and D. N. Card, "A Practical Ex- perience With Independent Verification and Validation," Proceedings of the Eighth International Computer Software and Applications Conference, November 1984

[3]Ramsey, J., and V. R. Basili, "Analyzing the Test Process Using Structural Coverage," Proceedings of the Eighth Inter- national Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

[4]Seidewitz, E., and M. Stark, "Towards a General Object- Oriented Software Development Methodology," Proceedings of the First International Symposium on Ada for the NASA Space Station, June 1986

Turner, C., and G. Caron, A Comparison of RADC and NASA/SEL Software Development Data, Data and Analysis Center for Software, Special Publication, May 1981

Turner, C., G. Caron, and G. Brement, NASA/SEL Data Compen- dium, Data and Analysis Center for Software, Special Publi- cation, April 1981

[3]Weiss, D. M., and V. R. Basili, "Evaluating Software De- velopment by Analysis of Changes: Some Data From the Soft- ware Engineering Laboratory," IEEE Transactions on Software Engineering, February 1985

[1]Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York: IEEE Computer Society Press, 1979

[2]Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," Empirical Foundations for Computer and Information Science (proceedings), November 1982

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977

NOTES:

[1]This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume I, July 1982.

[2]This article also appears in SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983.

[3]This article also appears in SEL-85-003, Collected Software Engineering Papers: Volume III, November 1985.

[4]This article also appears in SEL-86-004, Collected Software Engineering Papers: Volume IV, November 1986.